Abstract

This document describes an example of using communication (single-master transmission/reception) via the Renesas Serial Peripheral Interface (hereinafter called "RSPI") of the RZ/A1H to implement read and write access to the EEPROM.

Products

RZ/A1H

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.
Contents

1. Specifications ........................................................................................................................................ 4

2. Operation Confirmation Conditions .................................................................................................... 5

3. Reference Application Notes .............................................................................................................. 6

4. Hardware ............................................................................................................................................ 7
   4.1 Hardware Configuration ........................................................................................................... 7
   4.2 EEPROM Specification ............................................................................................................. 8
   4.3 Pins Used ................................................................................................................................. 9

5. Software ............................................................................................................................................. 10
   5.1 Operation Overview .................................................................................................................. 10
   5.1.1 Data Transfer to and from EEPROM Using SPI .............................................................. 10
   5.1.2 Accessing the EEPROM .................................................................................................. 12
   5.1.3 Operation Sequence .......................................................................................................... 13
   5.2 Peripheral Function Settings and Memory Allocation Used in Sample Code ....................... 15
   5.2.1 Peripheral Function Settings ............................................................................................ 15
   5.2.2 Section Assignment of Sample Code ............................................................................... 16
   5.3 Interrupts ..................................................................................................................................... 19
   5.4 Fixed-Width Integers .................................................................................................................. 19
   5.5 Constants ................................................................................................................................... 20
   5.6 Variables .................................................................................................................................... 22
   5.7 Functions .................................................................................................................................... 23
   5.8 Function Specifications ............................................................................................................... 27
   5.9 Flowcharts ................................................................................................................................... 57
   5.9.1 Main Processing ................................................................................................................... 57
   5.9.2 Sample Code Main Processing .......................................................................................... 58
   5.9.3 RSPI Sample Main Function ............................................................................................. 59
   5.9.4 Sample Function for Reading/Writing From/To EEPROM ............................................... 60
   5.9.5 PORT Initial Setting Function ........................................................................................... 61
   5.9.6 Sample Processing to Set or Release Protection for EEPROM ....................................... 62
   5.9.7 Sample Processing for Writing to EEPROM ................................................................. 63
   5.9.8 Sample Processing for Reading from EEPROM ............................................................. 63
   5.9.9 RSPI Channel 4 Transmit Buffer Empty Interrupt Processing ........................................ 64
   5.9.10 RSPI Channel 4 Receive Buffer Full Interrupt Processing ............................................ 64
   5.9.11 EEPROM Status Write Processing ............................................................................... 65
   5.9.12 EEPROM Status Read Processing .................................................................................. 65
   5.9.13 EEPROM Write-Enable Processing ................................................................................. 66
   5.9.14 Wait Processing for Completion of EEPROM Write Operation ................................... 66
   5.9.15 Processing for Data Transmission to EEPROM ............................................................ 67
   5.9.16 Processing for Data Reception from EEPROM .............................................................. 68
   5.9.17 RSPI Initial Settings ........................................................................................................... 69
   5.9.18 RSPI Function Enable Settings ........................................................................................ 70
   5.9.19 RSPI Function Disable Settings ....................................................................................... 71
   5.9.20 RSPI Transmit Processing .................................................................................................. 72
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.9.21</td>
<td>RSPI Receive Processing</td>
</tr>
<tr>
<td>5.9.22</td>
<td>RSPI Buffer Reset Processing</td>
</tr>
<tr>
<td>5.9.23</td>
<td>Wait Processing for RSPI End of Transmission</td>
</tr>
<tr>
<td>5.9.24</td>
<td>RSPI Transmit Buffer Empty Interrupt Notification Processing</td>
</tr>
<tr>
<td>5.9.25</td>
<td>RSPI Receive Buffer Full Interrupt Notification Processing</td>
</tr>
<tr>
<td>5.9.26</td>
<td>RSPI Channel 4 Initial Setting Function</td>
</tr>
<tr>
<td>5.9.27</td>
<td>Initialization Function for RSPI Channel 4 Transmit Buffer Empty Notification Information</td>
</tr>
<tr>
<td>5.9.28</td>
<td>Initialization Function for RSPI Channel 4 Receive Buffer Full Notification Information</td>
</tr>
<tr>
<td>5.9.29</td>
<td>Setting Function to Satisfy the Conditions for RSPI Channel 4 Transmit Buffer Empty Notification Information</td>
</tr>
<tr>
<td>5.9.30</td>
<td>Setting Function to Satisfy the Conditions for RSPI Channel 4 Receive Buffer Full Notification Information</td>
</tr>
<tr>
<td>5.9.31</td>
<td>Wait Function for Satisfaction of the Conditions for RSPI Channel 4 Transmit Buffer Empty Notification Information</td>
</tr>
<tr>
<td>5.9.32</td>
<td>Wait Function for Satisfaction of the Conditions for RSPI Channel 4 End of Transmission</td>
</tr>
<tr>
<td>5.9.33</td>
<td>Wait Function for Satisfaction of the Conditions for RSPI Channel 4 Receive Buffer Full Notification Information</td>
</tr>
<tr>
<td>5.9.34</td>
<td>Period Securing Function to Start Next Access RSPI Channel 4</td>
</tr>
<tr>
<td>5.10</td>
<td>Running the Sample Code</td>
</tr>
</tbody>
</table>

6. Sample Code

7. Reference Documents
1. Specifications

The RSPI is used to write/read data to/from the EEPROM. The RZ/A1H and the EEPROM operate as the master device and the slave device respectively.

In this application note, the interrupt controller, the serial communication interface with FIFO, the general I/O port, and the power-down mode are referred to as INTC, SCIF, PORT and STB respectively.

Table 1.1 lists the Peripheral Functions and Their Applications and Figure 1.1 shows the Operation Overview.

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSPI</td>
<td>RSPI channel 4 is used for reading from/writing to the EEPROM with the master transmit operation and the master receive operation in the serial peripheral interface communication using the RZ/A1H as the master device, and the EEPROM as the slave device.</td>
</tr>
<tr>
<td>STB</td>
<td>Used to supply a clock to RSPI channel 4</td>
</tr>
<tr>
<td>PORT</td>
<td>Used to switch the multiplexed pins of RSPI channel 4.</td>
</tr>
<tr>
<td>INTC</td>
<td>Used for RSPI interrupt control.</td>
</tr>
<tr>
<td>SCIF</td>
<td>Used in communication between the SCIF channel 2 and the host PC</td>
</tr>
</tbody>
</table>

Note: No EEPROM compatible with the SPI is mounted on the GENMAI board. To confirm the operation of this sample code, the customer should prepare a board to connect the EEPROM (R1EX25512ATA00I).
2. Operation Confirmation Conditions

The sample code accompanying this application note has been run and confirmed under the conditions below.

Table 2.1 Operation Confirmation Conditions

<table>
<thead>
<tr>
<th>Item</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU Used</td>
<td>RZ/A1H</td>
</tr>
<tr>
<td>Operating Frequency</td>
<td>CPU clock (I(\phi)): 400 MHz</td>
</tr>
<tr>
<td></td>
<td>Image processing clock (G(\phi)): 266.67 MHz</td>
</tr>
<tr>
<td></td>
<td>Internal bus clock (B(\phi)): 133.33 MHz</td>
</tr>
<tr>
<td></td>
<td>Peripheral clock 1 (P1(\phi)): 66.67 MHz</td>
</tr>
<tr>
<td></td>
<td>Peripheral clock 0 (P0(\phi)): 33.33 MHz</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>Power supply voltage (I/O): 3.3 V</td>
</tr>
<tr>
<td></td>
<td>Power supply voltage (internal): 1.18 V</td>
</tr>
<tr>
<td>Integrated development</td>
<td>ARM\textsuperscript{\textregistered} integrated development environment</td>
</tr>
<tr>
<td>environment</td>
<td>ARM Development Studio 5 (DS-5\textsuperscript{TM}) Version 5.16</td>
</tr>
<tr>
<td>C compiler</td>
<td>ARM C/C++ Compiler/Linker/Assembler Ver.5.03 [Build 102]</td>
</tr>
<tr>
<td>Compiler Option (excluding</td>
<td>-O3 -Ospace --cpu=Cortex-A9 --littleend --arm --apcs=/interwork/</td>
</tr>
<tr>
<td>additional directory path)</td>
<td>--no_unaligned_access --fpu=vfpv3_fp16 -g --asm</td>
</tr>
<tr>
<td>Operating Mode</td>
<td>Boot mode 0 (CS0 space 16-bit booting)</td>
</tr>
<tr>
<td>Terminal software</td>
<td>• Communication speed: 115200bps</td>
</tr>
<tr>
<td>communication settings</td>
<td>• Data length: 8 bits</td>
</tr>
<tr>
<td></td>
<td>• Parity: None</td>
</tr>
<tr>
<td></td>
<td>• Stop bit length: 1 bit</td>
</tr>
<tr>
<td></td>
<td>• Flow control: None</td>
</tr>
<tr>
<td>Board Used</td>
<td>GENMAI board</td>
</tr>
<tr>
<td>Device Used (Functions used</td>
<td>RTK772100BC00000BR (R7S72100 CPU board)</td>
</tr>
<tr>
<td>on the board)</td>
<td>• NOR flash memory (Connected to CS0 and CS1 spaces)</td>
</tr>
<tr>
<td></td>
<td>Manufacturer: Spansion Inc.</td>
</tr>
<tr>
<td></td>
<td>Product No.: S29GL512S10TFI01</td>
</tr>
<tr>
<td></td>
<td>• Serial interface (9-pin D-Sub connector)</td>
</tr>
<tr>
<td></td>
<td>• LED1</td>
</tr>
<tr>
<td>Device Used (Device not</td>
<td>EEPROM</td>
</tr>
<tr>
<td>mounted on the board)</td>
<td>Manufacturer: Renesas Electronics Corporation</td>
</tr>
<tr>
<td></td>
<td>Product No.: R1EX25512ATA00I</td>
</tr>
</tbody>
</table>

Notes: 1. The operating frequency used in clock mode 0 (Clock input of 13.33MHz from EXTAL pin)
2. No EEPROM compatible with the SPI is mounted on the GENMAI board. To confirm the operation of the sample code, the customer must connect the EEPROM to the board.
3. Reference Application Notes

For additional information associated with this document, refer to the following application notes.

- RZ/A1H Group I/O definition header file `<iodefine.h>` (R01AN1860EJ)
- RZ/A1H Group Example of Initialization (R01AN1864EJ)
4. Hardware

4.1 Hardware Configuration

Figure 4.1 shows the Connection Example in which the RZ/A1H is used as the master device and the EEPROM (R1EX25512ATA00I) as the slave device.

No EEPROM compatible with the SPI is mounted on the GENMAI board. To confirm the operation of this sample code, the customer should prepare the connection environment shown in Figure 4.1.

Notes:
1. The symbol # indicates negative logic (or active low).
2. The EEPROM (R1EX25512ATA00I) used in this application note requires time period to change the SSL signal to inactive mode. Therefore, the SSL signal is controlled by the RZ/A1H by connecting the SSL signal of the RZ/A1H to the S# pin of the EEPROM.

Figure 4.1 Connection Example
## 4.2 EEPROM Specification

Table 4.1 lists the EEPROM Specification.

<table>
<thead>
<tr>
<th>Item</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product No.</td>
<td>R1EX25512ATA00I</td>
</tr>
<tr>
<td>Power supply</td>
<td>Single power supply, 1.8 V to 5.5 V</td>
</tr>
<tr>
<td>Capacity</td>
<td>64KB (64K × 8 bits)</td>
</tr>
<tr>
<td>Transfer format</td>
<td>SPI mode 0 and SPI mode 3</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>5 MHz (2.5 V to 5.5 V), 3 MHz (1.8 V to 5.5 V)</td>
</tr>
<tr>
<td>Write operation</td>
<td>Page write function support (Support automatic incrementing for addresses (A6 to A0) within pages of up to 128 bytes)</td>
</tr>
<tr>
<td>Read operation</td>
<td>Sequential read function support (Enables continuous data output with incrementing the internal address register after 8 cycles of the serial input clock by maintaining low-level input on the S# pin.)</td>
</tr>
</tbody>
</table>
4.3 Pins Used

Table 4.2 lists the Pins Used and Their Functions.

Table 4.2 Pins Used and Their Functions

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Input/output</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A25 to A1</td>
<td>Output</td>
<td>Address signal output to the NOR flash memory</td>
</tr>
<tr>
<td>D15 to D0</td>
<td>Input/output</td>
<td>Data signal input/output of the NOR flash memory</td>
</tr>
<tr>
<td>CS0#</td>
<td>Output</td>
<td>Device select signal output to the NOR flash memory connected to CS0 space</td>
</tr>
<tr>
<td>RD#</td>
<td>Output</td>
<td>Read control signal output to the NOR flash memory</td>
</tr>
<tr>
<td>WE0#</td>
<td>Output</td>
<td>Write enable control signal output to the NOR flash memory</td>
</tr>
<tr>
<td>MD_BOOT1</td>
<td>Input</td>
<td>Selection of boot mode</td>
</tr>
<tr>
<td>MD_BOOT0</td>
<td>Input</td>
<td>MD_BOOT1: &quot;L&quot;, MD_BOOT0: &quot;L&quot; (Set to boot mode 0)</td>
</tr>
<tr>
<td>P4_10</td>
<td>Output</td>
<td>LED ON/OFF</td>
</tr>
<tr>
<td>RSPCK4</td>
<td>Output</td>
<td>Serial clock signal output to the EEPROM</td>
</tr>
<tr>
<td>SSL40</td>
<td>Output</td>
<td>Slave select signal output to the EEPROM</td>
</tr>
<tr>
<td>MOSI4</td>
<td>Output</td>
<td>Master outgoing data to the EEPROM</td>
</tr>
<tr>
<td>MISO4</td>
<td>Input</td>
<td>Slave outgoing data from the EEPROM</td>
</tr>
<tr>
<td>RxD2</td>
<td>Input</td>
<td>Serial receive data signal</td>
</tr>
<tr>
<td>TxD2</td>
<td>Output</td>
<td>Serial transmit data signal</td>
</tr>
</tbody>
</table>

Note: The symbol # indicates negative logic (or active low).
5. Software

5.1 Operation Overview

The peripheral functions (RSPI, PORT, STB, and INTC) are initialized to transmit/receive the data at the bit rate of 2.78Mbps using the RSPI channel 4. The 16KB of data is written to the EEPROM by using the SPI master transfer operation on the RSPI channel 4. The written data is then read from the EEPROM.

5.1.1 Data Transfer to and from EEPROM Using SPI

The SPI master transfer operation is used to write/read 16KB of data to/from the EEPROM.

In the EEPROM, the data is imported at the rising edge and changed at the falling edge of the RSPCK clock. Therefore, the transfer format is provided as follows; the CPHA and the CPOL bits in the command register (SPCMD) are both set to "1", the MOSI data output is changed at the falling edge of the RSPCK clock, and the MISO data is sampled at the rising edge of the RSPCK clock.

The RSPI is specified for the 8-bit transfer data length and the MSB first transfer as the data transfer format.

Figure 5.1 shows the Timing Example for Data Transfers with EEPROM Using SPI.

In the sample code, the RSPI is set to satisfy the timing conditions listed in Table 5.1 and Table 5.2 when the CPHA bit and the CPOL bit are both set to "1".

![Figure 5.1 Timing Example for Data Transfers with EEPROM Using SPI](image-url)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{\text{SPyc}} )</td>
<td>RSPCK clock size</td>
<td>Bit rate for SPI transfer. This bit rate is set to approximately 359ns (2.78 MHz) in the sample code.</td>
</tr>
</tbody>
</table>
| \( t_{SU} \) | Data input setup time | Time required from data output by the slave to data sampling. This is set to satisfy the following formula: 
\[
1/2 \times t_{\text{SPyc}} - t_{\text{CLQV}}(\text{MAX}) > t_{SU}(\text{MIN})
\] |
| \( t_h \) | Data input hold time | Time required to stop data output by the slave after data sampling. This is set to satisfy the following formula: 
\[
t_{\text{CLQX}}(\text{MIN}) + 1/2 \times t_{\text{SPyc}} > t_h(\text{MIN})
\] |
| \( t_{\text{LEAD}} \) | SSL setup time | Time from the start of SSL signal assertion to the RSPCK oscillation. |
| \( t_{\text{LAG}} \) | SSL hold time | Time until the SSL signal is negated after sending the last RSPCK edge of the SPI transfer. |
| \( t_{OD} \) | Data output delay time | Time until the master outputs the data after the falling edge of the RSPCK clock. |
| \( t_{OH} \) | Data input hold time | Time that the data output is held by the master after the falling edge of the RSPCK clock. |
| \( t_{TD} \) | Continuous transmission delay time | Inactive period of the SSL signal from the end of SPI transfer to the next SPI transfer. |

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
</table>
| \( t_{\text{SLCH}} \) | Chip select “L” setup time | Time required for the EEPROM to sample the data at the rising edge of the RSPCK clock after the SSL assertion. This is set to satisfy the following formula: 
\[
t_{\text{TD}}(\text{MIN}) > t_{\text{SLCH}}(\text{MIN})
\] |
| \( t_{\text{SHSL}} \) | Chip select “H” time | Time required as the SSL negation period. This is set to satisfy the following formula: 
\[
t_{\text{TD}}(\text{MIN}) > t_{\text{SHSL}}(\text{MIN})
\] |
| \( f_c \) | Serial clock frequency | EEPROM-corresponderable maximum operating frequency. This is set to satisfy the following formula: 
\[
f_c(\text{MAX}) > 1/t_{\text{SPyc}}
\] |
| \( t_{\text{CHSH}} \) | Chip select “L” hold time | Holding time required to negate the SSL signal after the last rising edge of the RSPCK clock. This is set to satisfy the following formula: 
\[
t_{\text{LAG}}(\text{MIN}) > t_{\text{CHSH}}(\text{MIN})
\] |
| \( t_{\text{DVCH}} \) | Data input setup time | Time required from data output by the master to data sampling. This is set to satisfy the following formula: 
\[
1/2 \times t_{\text{SPyc}} - t_{\text{OD}}(\text{MAX}) > t_{\text{DVCH}}(\text{MIN})
\] |
| \( t_{\text{CHDX}} \) | Data input hold time | Time required to stop data output by the master after data sampling. This is set to satisfy the following formula: 
\[
t_{\text{OH}}(\text{MIN}) + 1/2 \times t_{\text{SPyc}} > t_{\text{CHDX}}(\text{MIN})
\] |
| \( t_{\text{CLOV}} \) | Data output delay time | Time that the EEPROM outputs the data after the falling edge of the RSPCK clock. |
| \( t_{\text{CLQX}} \) | Data output hold time | Time that the EEPROM holds the data output after the falling edge of the RSPCK clock. |
5.1.2 Accessing the EEPROM

The instruction codes are used to access the EEPROM. Table 5.3 lists the Instruction Code Sets of EEPROM (R1EX25512ATA001).

Table 5.3 Instruction Code Sets of EEPROM (R1EX25512ATA001)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Address Byte Count</th>
<th>Data Byte Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREN</td>
<td>H'06</td>
<td>0</td>
<td>0</td>
<td>Write enable</td>
</tr>
<tr>
<td>WRDI</td>
<td>H'04</td>
<td>0</td>
<td>0</td>
<td>Write disable</td>
</tr>
<tr>
<td>RDSR</td>
<td>H'05</td>
<td>0</td>
<td>1</td>
<td>Read status register</td>
</tr>
<tr>
<td>WRSR</td>
<td>H'01</td>
<td>0</td>
<td>1</td>
<td>Write status register</td>
</tr>
<tr>
<td>READ</td>
<td>H'03</td>
<td>2</td>
<td>1 or more*1</td>
<td>Read memory</td>
</tr>
<tr>
<td>WRITE</td>
<td>H'02</td>
<td>2</td>
<td>1 to 128*2</td>
<td>Write memory</td>
</tr>
</tbody>
</table>

Notes:
1. When S# is held to "L", the read operations cover an area staring from the specified address, which is incremented automatically. After the most significant address of the EEPROM is reached, the address returns to H'0.
2. When S# is held to "L", the write address is incremented automatically within the area on the same page as the specified address. If the end of the page is reached, the address returns to the top address of the page.

Figure 5.2 shows the READ Command Sequence Example when the instruction code set is used.

The READ command transmits the 2-byte address (A15 to A0) subsequently to the instruction code (H'03) after asserting the SSL signal. Then the read data is transferred from the slave EEPROM at every falling edge of the RSPCK.

Regarding other commands except the READ command, they also access to the EEPROM by specifying the address and the data subsequent to the instruction codes listed in Table 5.3. (The commands with 0 byte for address and data do not require a transfer sequence for address or data.) Before transmitting the WRITE and the WRSR commands, the WREN command should be transmitted to set the "Write Enable Latch (WEL)" bit to "1".

Figure 5.2 READ Command Sequence Example
5.1.3 Operation Sequence

Figure 5.3 and Figure 5.4 show the Operation Sequence of Sample Code as an example for write/read operations to/from the EEPROM.

Sets PORT multiplex functions used for transmission/reception on RSPI channel 4.
Sets STB to enable the RSPI channel 4.
Sets the bit rate to 2.78 Mbps on RSPI channel 4.
Sets INTC to enable the interrupts on RSPI channel 4.

Transmits instruction code "WREN (H'06)".
Transmits instruction code "WRSR (H'01)".
Writes "H'00" to the status register to set the "Block Protect Bits (BP1 and BP0)" to B'00 and to release protection on the entire EEPROM area.

Transmits instruction code "RDSR (H'05)":
Reads the status register repeatedly until the value of the "Write In Progress Bits (WIP)" changes to B'0 (no write in progress).

Write operations are performed repeatedly until 16KB of data has been written to the EEPROM.
Transmits instruction code "WREN (H'06)".
Transmits instruction code "WRITE (H'02)".
Transmits a 2-byte memory address.
Writes 128 bytes of write data to the EEPROM.

Transmits instruction code "RDSR (H'05)"
Reads the status register repeatedly until the value of the "Write In Progress Bits (WIP)" bit changes to B'0 (no write in progress).

Figure 5.3 Operation Sequence of Sample Code (1/2)
Read operation from EEPROM

- Transmits instruction code "READ (H'03)".
- Transmits a 2-byte memory address.
- Reads 16KB of data from the EEPROM.

Protection setting to EEPROM

- Transmits instruction code "WREN (H'06)".
- Transmits instruction code "WRSR (H'01)".
- Writes "H'0C" to the status register to set the "Block Protect Bits (BP1 and BP0)" to B'11 and set protection for the entire EEPROM area.
- Transmits instruction code "RDSR (H'05)".
- Reads the status register repeatedly until the value of the "Write In Progress Bits (WIP)" changes to B'0 (no write in progress).

Figure 5.4  Operation Sequence of Sample Code (2/2)
## 5.2 Peripheral Function Settings and Memory Allocation Used in Sample Code

### 5.2.1 Peripheral Function Settings

Table 5.4 lists the settings when the sample code is executed.

Table 5.4 Peripheral Function Settings

<table>
<thead>
<tr>
<th>Module</th>
<th>Setting</th>
</tr>
</thead>
</table>
| RSPI   | Channel 4  
|        | Master/slave: Single master  
|        | Bit rate: 2.78Mbps (Dividing ratio is set to 24 when P1\(f\) is 66.67MHz\(^*\))  
|        | SSL polarity: 0-active  
|        | MOSI idle value: Fixed to 1  
|        | RSPCK delay: 1 RSPCK  
|        | SSL negation delay: 1 RSPCK  
|        | Next access delay: 1 RSPCK + 2P1\(f\)  
|        | Access width: Byte access to data register (SPDR)  
|        | Sequence operation: Referenced SPCMD register 0 \(\rightarrow 0 \rightarrow \ldots\)  
|        | Data format: MSB first  
|        | Transfer data length: 8 bits  
|        | SSL signal: SSL signal level is held from the end of transfer to the start of next access.  
|        | RSPCK polarity: When RSPCK in idle state is 1 (CPOL = 1)  
|        | RSPCK phase: Data change at odd edge, data sampling at even edge (CPHA = 1)  
|        | Dummy data transmission: Disabled  
|        | Transmit buffer data triggering number: 0 byte or less  
|        | Receive buffer data triggering number: 24 bytes or more  
|        | Error interrupt request generation: Disabled  
|        | Mode fault error detection: Disabled  
|        | Loopback: Normal mode  
| STB    | Clock is supplied to the RSPI channel 4.  
| PORT   | PORT4 multiplexed pin function settings  
|        | P4_0: RSPCK4, P4_1: SSL40  
|        | P4_2: MOSI4, P4_3: MISO4  
| INTC   | Registration and execution of handlers for the RSPI channel 4 receive buffer full interrupt (interrupt ID: 283) and the RSPI channel 4 transfer data empty interrupt (interrupt ID: 284)  
| SCIF   | Sets the channel 2 in asynchronous communication mode.  
|        | Data length: 8 bits  
|        | Stop bits: 1 bit  
|        | Parity: None  
|        | Sets the clock source without frequency dividing and the bit rate value at 17. Sets the bit rate to be 115200bps when P1\(f\) is 66.67MHz.  
|        | Difference is 0.46%  

Note: \(^*\) Refer to RZ/A1H Group User's Manual for the formula used to calculate the bit rate.
5.2.2 Section Assignment of Sample Code

Table 5.5 and Table 5.6 list the Sections Used in the sample code, and Figure 5.5 shows the sample code section assignment in the initial state (load view) and the one after the scatter loading function is used (execution view).

For details about the sections and the scatter loading function, refer to "Image structure and generation" in ARM Compiler toolchain: Using the Linker.

### Table 5.5 Sections Used (1/2)

<table>
<thead>
<tr>
<th>Area Name</th>
<th>Description</th>
<th>Type</th>
<th>Load Area</th>
<th>Execution Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>VECTOR_TABLE</td>
<td>Exception processing vector table</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td>RESET_HANDLER</td>
<td>Program code area of reset handler processing</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td></td>
<td>This area consists of the following sections.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* INITCA9CACHE (L1 cache setting)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* INIT_TTB (MMU setting)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* RESET_HANDLER (Reset handler)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE_BASIC_SETUP</td>
<td>Program code area to optimize operating frequency and flash memory</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td>InRoot</td>
<td>This area consists of the sections located in the root area such as C standard library.</td>
<td>Code</td>
<td>and RO Data</td>
<td>FLASH</td>
</tr>
<tr>
<td>CODE_FPU_INIT</td>
<td>Program code area for NEON and VFP initializations</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td></td>
<td>This area consists of the following sections.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* CODE_FPU_INIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* FPU_INIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE_RESET</td>
<td>Program code area for hardware initialization</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td></td>
<td>This area consists of the following sections.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* CODE_RESET (Startup processing)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* INIT_VBAR (Vector base setting)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE_IO_REGRW</td>
<td>Program code area for read/write functions of I/O register</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>CODE</td>
<td>Program code area for defaults</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td></td>
<td>All the Code type sections which do not define section names with C source are assigned in this area.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONST</td>
<td>Constant data area for defaults</td>
<td>RO Data</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
<tr>
<td>Area Name</td>
<td>Description</td>
<td>Type</td>
<td>Load Area</td>
<td>Execution Area</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
<td>-----------</td>
<td>----------------</td>
</tr>
<tr>
<td>VECTOR_MIRROR_TABLE</td>
<td>Exception processing vector table (Section to transfer data to large-capacity on-chip RAM)</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>CODE_HANDLER_JMPTBL</td>
<td>Program code area for user-defined functions of IRQ interrupt handler</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>CODE_HANDLER</td>
<td>Program code area of IRQ interrupt handler</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td></td>
<td>This area consists of the following sections.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CODE_HANDLER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• IRQ_FIQ_HANDLER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE_CACHEOPERATION</td>
<td>Program code area for setting the L1 and L2 caches*3</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>DATA_HANDLER_JMPTBL</td>
<td>Registration table data area for user-defined functions of IRQ interrupt handler</td>
<td>RW Data</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>ARM_LIB_STACK</td>
<td>Application stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>IRQ_STACK</td>
<td>IRQ mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>FIQ_STACK</td>
<td>FIQ mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>SVC_STACK</td>
<td>Supervisor (SVC) mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>ABT_STACK</td>
<td>Abort (ABT) mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>TTB</td>
<td>MMU translation table area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>ARM_LIB_HEAP</td>
<td>Application heap area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>DATA</td>
<td>Data area with initial value for defaults</td>
<td>RW Data</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td></td>
<td>All the RW Data type sections which do not define section names with C source are assigned in this area.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSS</td>
<td>Data area without initial value for defaults</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td></td>
<td>All the ZI Data type sections which do not define section names with C source area assigned in this area.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSS_RSPI_SAMPLE</td>
<td>Data area without initial value used by RSPI sample code</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td></td>
<td>Defined to be assigned to the L1 cache disable area.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. In the listing of load areas and execution areas in the table, “FLASH” indicates the NOR flash memory area and “LRAM” the large-capacity on-chip RAM area.
2. The section names basically are the same as the area names, but the RESET_HANDLER, InRoot, CODE_FPU_INIT, CODE_RESET, CODE, CONST, CODE_HANDLER, DATA, and BSS areas each comprise multiple sections. For details on areas and sections, refer to the ARM compiler toolchain manual.
3. This section should be placed in the cache-disabled area.
Cache-disabled space in large-capacity on-chip RAM (10MB)

H'FFFF FFFF
H'6090 0000
H'6000 0000

CS0 space (64 MB)

H'2000 0000
H'1C00 0000
H'1800 0000
H'1000 0000

Large-capacity on-chip RAM (10MB)

H'6000 0000
H'60A0 0000

Section assignment (load view)

Memory allocation after scatter loading

Section assignment (execution view)

Transfer cache operation process to cache-disabled area in on-chip RAM

Clear to 0

Initialize data with initial values

Obtain areas for stacks and the like

Initialize data with initial values

Transfer program code that requires high-speed processing to the on-chip RAM

Transfer exception processing vector to the on-chip RAM

Figure 5.5 Section Assignment
5.3 Interrupts

Table 5.7 lists the Interrupt Used in Sample Code.

Table 5.7  Interrupt Used in Sample Code

<table>
<thead>
<tr>
<th>Interrupt Source (Interrupt ID)</th>
<th>Priority</th>
<th>Processing Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPRI4 (283)</td>
<td>10</td>
<td>Notifies that the RSPI receive buffer has received the data equal to or more than the byte count specified by the receive buffer data triggering number (RXTRG bits) from the EEPROM.</td>
</tr>
<tr>
<td>SPTI4 (284)</td>
<td>10</td>
<td>Notifies that the data has been transmitted to the EEPROM and the data in the RSPI transmit buffer is equal to or smaller than the byte count specified by the transmit buffer data triggering number (TXTRG bits).</td>
</tr>
</tbody>
</table>

5.4 Fixed-Width Integers

Table 5.8 lists the Fixed-Width Integers Used in Sample Code.

Table 5.8  Fixed-Width Integers Used in Sample Code

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char_t</td>
<td>8-bit character</td>
</tr>
<tr>
<td>bool_t</td>
<td>Boolean type, value: true (1) or false (0)</td>
</tr>
<tr>
<td>int_t</td>
<td>High-speed integer, signed 32-bit integer in this sample code</td>
</tr>
<tr>
<td>int8_t</td>
<td>8-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td>int16_t</td>
<td>16-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td>int32_t</td>
<td>32-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td>int64_t</td>
<td>64-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td>uint8_t</td>
<td>8-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td>uint16_t</td>
<td>16-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td>uint32_t</td>
<td>32-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td>uint64_t</td>
<td>64-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td>float32_t</td>
<td>32-bit floating point (Defined by standard library when specifying &quot;<strong>ARM_NEON</strong>&quot;)</td>
</tr>
<tr>
<td>float64_t</td>
<td>64-bit floating point (Defined by standard library when specifying &quot;<strong>ARM_NEON</strong>&quot;)</td>
</tr>
<tr>
<td>float128_t</td>
<td>128-bit floating point</td>
</tr>
</tbody>
</table>
5.5 Constants

Table 5.9 Table 5.10 list the Constants Used in Sample Code..

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMPLE_RSPI_REQ_PROTECT</td>
<td>(0x00)</td>
<td>EEPROM protection request</td>
</tr>
<tr>
<td>SAMPLE_RSPI_REQ_UNPROTECT</td>
<td>(0x01)</td>
<td>EEPROM protection release request</td>
</tr>
<tr>
<td>EEP_01BYTE_READ</td>
<td>(1)</td>
<td>Setting for 1-byte data reception</td>
</tr>
<tr>
<td>EEP_nBYTE_READ</td>
<td>(0)</td>
<td>Setting for data reception for 1 byte or more</td>
</tr>
<tr>
<td>EEP_CMD_WRITE_ENABLE</td>
<td>(0x06u)</td>
<td>Write-enable</td>
</tr>
<tr>
<td>EEP_CMD_WRITE_DISABLE</td>
<td>(0x04u)</td>
<td>Write-disable</td>
</tr>
<tr>
<td>EEP_CMD_READ_STATUS</td>
<td>(0x05u)</td>
<td>Read from status register</td>
</tr>
<tr>
<td>EEP_CMD_READ_ARRAY</td>
<td>(0x01u)</td>
<td>Read from memory</td>
</tr>
<tr>
<td>EEP_CMD_WRITE_ARRAY</td>
<td>(0x02u)</td>
<td>Write to memory</td>
</tr>
<tr>
<td>EEP_STATUS_UNPROTECT</td>
<td>(0x00u)</td>
<td>Setting values of EEPROM status register</td>
</tr>
<tr>
<td>EEP_STATUS_PROTECT</td>
<td>(0xCu)</td>
<td>Protection setting</td>
</tr>
<tr>
<td>EEP_STATUS_BUSY</td>
<td>(0x01u)</td>
<td>Mask value of the WIP bit to confirm write operation of the EEPROM status register</td>
</tr>
<tr>
<td>RSPI_EEP_TOP_ADDRESS</td>
<td>(0)</td>
<td>Start address of the EEPROM to be accessed</td>
</tr>
<tr>
<td>RSPI_EEP_PAGE_SIZE</td>
<td>(128)</td>
<td>EEPROM page size</td>
</tr>
<tr>
<td>RSPI_EEP_DATA_SIZE</td>
<td>(0x4000)</td>
<td>EEPROM data size (16KB) where the sample code is read/written</td>
</tr>
<tr>
<td>RSPI_SPDR_ACCESS_8</td>
<td>(1)</td>
<td>Access widths of transmit/receive buffers</td>
</tr>
<tr>
<td>RSPI_SPDR_ACCESS_16</td>
<td>(2)</td>
<td>8-bit access</td>
</tr>
<tr>
<td>RSPI_SPDR_ACCESS_32</td>
<td>(3)</td>
<td>16-bit access</td>
</tr>
<tr>
<td>RSPI_TXTRG_CURRENT</td>
<td>(0xFF)</td>
<td>Transmit buffer data triggering number (Setting values of the TXTRG bits)</td>
</tr>
<tr>
<td>RSPI_TXTRG_07_BYTE</td>
<td>(0)</td>
<td>RSPI_TXTRG_CURRENT: Current setting value is used.</td>
</tr>
<tr>
<td>RSPI_TXTRG_06_BYTE</td>
<td>(1)</td>
<td>RSPI_TXTRG_07_BYTE: Change to 7 bytes</td>
</tr>
<tr>
<td>RSPI_TXTRG_04_BYTE</td>
<td>(2)</td>
<td>RSPI_TXTRG_06_BYTE: Change to 6 bytes</td>
</tr>
<tr>
<td>RSPI_TXTRG_00_BYTE</td>
<td>(3)</td>
<td>RSPI_TXTRG_04_BYTE: Change to 4 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSPI_TXTRG_00_BYTE: Change to 0 byte</td>
</tr>
</tbody>
</table>
Table 5.10 Constants Used in Sample Code (2/2)

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSPI_RXTRG_CURRENT</td>
<td>(0xFF)</td>
<td>Receive buffer data triggering number (Setting values of the RXTRG bits)</td>
</tr>
<tr>
<td>RSPI_RXTRG_01_BYTE</td>
<td>(0)</td>
<td>RSPI_RXTRG_CURRENT: Current setting value is used</td>
</tr>
<tr>
<td>RSPI_RXTRG_02_BYTE</td>
<td>(1)</td>
<td>RSPI_RXTRG_01_BYTE: Change to 1 byte</td>
</tr>
<tr>
<td>RSPI_RXTRG_04_BYTE</td>
<td>(2)</td>
<td>RSPI_RXTRG_02_BYTE: Change to 2 bytes</td>
</tr>
<tr>
<td>RSPI_RXTRG_08_BYTE</td>
<td>(3)</td>
<td>RSPI_RXTRG_04_BYTE: Change to 4 bytes</td>
</tr>
<tr>
<td>RSPI_RXTRG_16_BYTE</td>
<td>(4)</td>
<td>RSPI_RXTRG_08_BYTE: Change to 8 bytes</td>
</tr>
<tr>
<td>RSPI_RXTRG_24_BYTE</td>
<td>(5)</td>
<td>RSPI_RXTRG_16 BYTE: Change to 16 bytes</td>
</tr>
<tr>
<td>RSPI_RXTRG_32_BYTE</td>
<td>(6)</td>
<td>RSPI_RXTRG_24_BYTE: Change to 24 bytes</td>
</tr>
<tr>
<td>RSPI_RXTRG_05_BYTE</td>
<td>(7)</td>
<td>RSPI_RXTRG_32_BYTE: Change to 32 bytes</td>
</tr>
<tr>
<td>RSPI_SPR</td>
<td>(5)</td>
<td>Value set to the SPBR register</td>
</tr>
<tr>
<td>RSPI_BRDV</td>
<td>(1)</td>
<td>Setting value for BRDV1 and BRDV0 bits in the SPCMD0 register (Bits BRDV[1:0] are set to B'01.) In the sample code, the BRDV bits in the SPBR register are used to specify the division ratio for divide-by-24, thereby the RSPI channel 4 bit rate is set to P1f/24 (= 2.78Mbps).</td>
</tr>
</tbody>
</table>
5.6 Variables
Table 5.11 lists the static Variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>static volatile</td>
<td>rspi_receive_full_flg</td>
<td>RSPI Receive buffer full interrupt notification flag</td>
<td>Userdef_RSPI4_InitReceiveFull Userdef_RSPI4_SetReceiveFull Userdef_RSPI4_WaitReceiveFull</td>
</tr>
<tr>
<td>uint8_t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static volatile</td>
<td>rspi_transmit_empty_flg</td>
<td>RSPI transmit buffer empty interrupt notification flag</td>
<td>Userdef_RSPI4_InitTransmitEmpty Userdef_RSPI4_SetTransmitEmpty Userdef_RSPI4_WaitTransmitEmpty</td>
</tr>
<tr>
<td>uint8_t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static uint8_t</td>
<td>rspi_trans_data[RSPI_EEP_DATA_SIZE]</td>
<td>Transmit data buffer*</td>
<td>Sample_RSPI_EepAccess</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static uint8_t</td>
<td>rspi_receive_data[RSPI_EEP_DATA_SIZE]</td>
<td>Receive data buffer*</td>
<td>Sample_RSPI_EepAccess</td>
</tr>
</tbody>
</table>

Note: rspi_trans_data and rspi_receive_data are defined as the BSS_RSPI_SAMPLE section and assigned to the mirror space (H'6000 0000 to H'609F FFFF) in the large-capacity on-chip RAM. This space disables the L1 cache by setting the MMU.

Figure 5.6 shows the Memory Allocation for Transmit Data Buffer and Receive Data Buffer.

![Figure 5.6 Memory Allocation for Transmit Data Buffer and Receive Data Buffer](image-url)
5.7 Functions

The sample code consists of the interface functions (API functions) to perform RSPI transfers and the like, the user-defined functions (functions called by the API functions) which should be prepared by the user corresponding to the system applied, and the sample functions required to operate the sample code such as read/write access to the EEPROM.

Table 5.12 and Table 5.13 list the Sample Functions and the API Functions respectively.

Table 5.14 to Table 5.16 list the Use-defined Functions.

Table 5.12 Sample Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Main processing</td>
</tr>
<tr>
<td>Sample_Main</td>
<td>Sample code main processing</td>
</tr>
<tr>
<td>Sample_RSPI_Main</td>
<td>RSPI sample code main processing</td>
</tr>
<tr>
<td>Sample_RSPI_EepAccess</td>
<td>Sample processing for reading/writing from/to EEPROM</td>
</tr>
<tr>
<td>RSPI_PORT_Init</td>
<td>Initial setting of PORT used in RSPI transmission/reception</td>
</tr>
<tr>
<td>Sample_RSPI_EepControlProtect_8bit</td>
<td>Sample processing to set or release protection for EEPROM</td>
</tr>
<tr>
<td>Sample_RSPI_EepWrite_8bit</td>
<td>Sample processing for writing to EEPROM</td>
</tr>
<tr>
<td>Sample_RSPI_EepRead_8bit</td>
<td>Sample processing for reading from EEPROM</td>
</tr>
<tr>
<td>Sample_RSPI_Spti4_Interrupt</td>
<td>RSPI channel 4 transmit buffer empty interrupt processing</td>
</tr>
<tr>
<td>Sample_RSPI_Spri4_Interrupt</td>
<td>RSPI channel 4 receive buffer full interrupt processing</td>
</tr>
<tr>
<td>RSPI_EepWriteStatus_8bit</td>
<td>EEPROM status write processing</td>
</tr>
<tr>
<td>RSPI_EepReadStatus_8bit</td>
<td>EEPROM status read processing</td>
</tr>
<tr>
<td>RSPI_EepWriteEnable_8bit</td>
<td>EEPROM write-enable processing</td>
</tr>
<tr>
<td>RSPI_EepBusyWait_8bit</td>
<td>Waiting for completion of EEPROM write operation</td>
</tr>
<tr>
<td>RSPI_EepWrite</td>
<td>Processing for data transmission to EEPROM</td>
</tr>
<tr>
<td>RSPI_EepRead</td>
<td>Processing for data reception from EEPROM</td>
</tr>
</tbody>
</table>

Table 5.13 API Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_RSPI_Init</td>
<td>RSPI initial setting</td>
</tr>
<tr>
<td>R_RSPI_Open</td>
<td>RSPI function enable setting</td>
</tr>
<tr>
<td>R_RSPI_Close</td>
<td>RSPI function disable setting</td>
</tr>
<tr>
<td>R_RSPI_Write</td>
<td>RSPI transmit processing</td>
</tr>
<tr>
<td>R_RSPI_Read</td>
<td>RSPI receive processing</td>
</tr>
<tr>
<td>R_RSPI_ResetBuffer</td>
<td>RSPI buffer reset processing</td>
</tr>
<tr>
<td>R_RSPI_WaitTransmitEnd</td>
<td>Waiting for RSPI end of transmission</td>
</tr>
<tr>
<td>R_RSPI_SptiInterrupt</td>
<td>RSPI transmit buffer empty interrupt notification processing</td>
</tr>
<tr>
<td>R_RSPI_SpriInterrupt</td>
<td>RSPI receive buffer full interrupt notification processing</td>
</tr>
<tr>
<td>Function Name</td>
<td>Outline</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Userdef_RSP0_Init             | RSPI channel 0 initial setting  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP1_Init             | RSPI channel 1 initial setting  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP2_Init             | RSPI channel 2 initial setting  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP3_Init             | RSPI channel 3 initial setting  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP4_Init             | RSPI channel 4 initial setting  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP0_InitTransmitEmpty| Initialization of RSPI channel 0 transmit buffer empty notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP1_InitTransmitEmpty| Initialization of RSPI channel 1 transmit buffer empty notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP2_InitTransmitEmpty| Initialization of RSPI channel 2 transmit buffer empty notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP3_InitTransmitEmpty| Initialization of RSPI channel 3 transmit buffer empty notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP4_InitTransmitEmpty| Initialization of RSPI channel 4 transmit buffer empty notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP0_InitReceiveFull   | Initialization of RSPI channel 0 receive buffer full notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP1_InitReceiveFull   | Initialization of RSPI channel 1 receive buffer full notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP2_InitReceiveFull   | Initialization of RSPI channel 2 receive buffer full notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP3_InitReceiveFull   | Initialization of RSPI channel 3 receive buffer full notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP4_InitReceiveFull   | Initialization of RSPI channel 4 receive buffer full notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP0_SetTransmitEmpty  | Setting to satisfy the conditions for RSPI channel 0 transmit buffer empty notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP1_SetTransmitEmpty  | Setting to satisfy the conditions for RSPI channel 1 transmit buffer empty notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP2_SetTransmitEmpty  | Setting to satisfy the conditions for RSPI channel 2 transmit buffer empty notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP3_SetTransmitEmpty  | Setting to satisfy the conditions for RSPI channel 3 transmit buffer empty notification  
information  
(In the sample code, this function returns without any processing.) |
| Userdef_RSP4_SetTransmitEmpty  | Setting to satisfy the conditions for RSPI channel 4 transmit buffer empty notification  
information  
(In the sample code, this function returns without any processing.) |
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Userdef_RSPI0_SetReceiveFull</td>
<td>Setting to satisfy the conditions for RSPI channel 0 receive buffer notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI1_SetReceiveFull</td>
<td>Setting to satisfy the conditions for RSPI channel 1 receive buffer notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI2_SetReceiveFull</td>
<td>Setting to satisfy the conditions for RSPI channel 2 receive buffer notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI3_SetReceiveFull</td>
<td>Setting to satisfy the conditions for RSPI channel 3 receive buffer notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI4_SetReceiveFull</td>
<td>Setting to satisfy the conditions for RSPI channel 4 receive buffer notification information</td>
</tr>
<tr>
<td>Userdef_RSPI0_WaitTransmitEmpty</td>
<td>Waiting for satisfaction of the conditions for RSPI channel 0 transmit buffer empty notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI1_WaitTransmitEmpty</td>
<td>Waiting for satisfaction of the conditions for RSPI channel 1 transmit buffer empty notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI2_WaitTransmitEmpty</td>
<td>Waiting for satisfaction of the conditions for RSPI channel 2 transmit buffer empty notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI3_WaitTransmitEmpty</td>
<td>Waiting for satisfaction of the conditions for RSPI channel 3 transmit buffer empty notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI4_WaitTransmitEmpty</td>
<td>Waiting for satisfaction of the conditions for RSPI channel 4 transmit buffer empty notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI0_WaitTransmitEnd</td>
<td>Waiting for RSPI channel 0 end of transmission (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI1_WaitTransmitEnd</td>
<td>Waiting for RSPI channel 1 end of transmission (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI2_WaitTransmitEnd</td>
<td>Waiting for RSPI channel 2 end of transmission (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI3_WaitTransmitEnd</td>
<td>Waiting for RSPI channel 3 end of transmission (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI4_WaitTransmitEnd</td>
<td>Waiting for RSPI channel 4 end of transmission</td>
</tr>
<tr>
<td>Userdef_RSPI0_WaitReceiveFull</td>
<td>Waiting for satisfaction of the conditions for RSPI channel 0 receive buffer full notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI1_WaitReceiveFull</td>
<td>Waiting for satisfaction of the conditions for RSPI channel 1 receive buffer full notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI2_WaitReceiveFull</td>
<td>Waiting for satisfaction of the conditions for RSPI channel 2 receive buffer full notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI3_WaitReceiveFull</td>
<td>Waiting for satisfaction of the conditions for RSPI channel 3 receive buffer full notification information (In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPI4_WaitReceiveFull</td>
<td>Waiting for satisfaction of the conditions for RSPI channel 4 receive buffer full notification information</td>
</tr>
</tbody>
</table>
Table 5.16  User-Defined Functions (3/3)

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Userdef_RSPi0_DelayNextAccess</td>
<td>Securing period to start next access on RSPI channel 0</td>
</tr>
<tr>
<td></td>
<td>(In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPi1_DelayNextAccess</td>
<td>Securing period to start next access on RSPI channel 1</td>
</tr>
<tr>
<td></td>
<td>(In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPi2_DelayNextAccess</td>
<td>Securing period to start next access on RSPI channel 2</td>
</tr>
<tr>
<td></td>
<td>(In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPi3_DelayNextAccess</td>
<td>Securing period to start next access on RSPI channel 3</td>
</tr>
<tr>
<td></td>
<td>(In the sample code, this function returns without any processing.)</td>
</tr>
<tr>
<td>Userdef_RSPi4_DelayNextAccess</td>
<td>Securing period to start next access on RSPI channel 4</td>
</tr>
</tbody>
</table>
5.8 Function Specifications

The following tables list the sample code function specifications.

### main

<table>
<thead>
<tr>
<th>Outline</th>
<th>Main processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>int_t main(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Displays the sample code information to the terminal software running on the host PC connected to the GENMAI board via the serial interface.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>0</td>
</tr>
</tbody>
</table>

### Sample_Main

<table>
<thead>
<tr>
<th>Outline</th>
<th>Sample code main processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>void Sample_Main(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Waits for character input from the terminal software running on the host PC connected to the GENMAI board via the serial interface. Activates the RSPI sample code when &quot;RSPI&quot; + &quot;Enter&quot; key is entered from the terminal window.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

### Sample_RSPI_Main

<table>
<thead>
<tr>
<th>Outline</th>
<th>RSPI sample code main processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>int32_t Sample_RSPI_Main(int32_t argc, char_t **argv)</td>
</tr>
<tr>
<td>Description</td>
<td>Waits for character input from the terminal software running on the host PC connected to the GENMAI board via the serial interface. The sample code is executed when &quot;1&quot; + &quot;Enter&quot; key is entered</td>
</tr>
<tr>
<td>Arguments</td>
<td>int32_t argc : Number of command arguments input from terminal</td>
</tr>
<tr>
<td></td>
<td>char_t **argv : Pointer to command input from terminal</td>
</tr>
<tr>
<td>Return Value</td>
<td>COMMAND_EXIT: Termination of RSPI sample code processing</td>
</tr>
</tbody>
</table>
### Sample_RSPI_EepAccess

**Outline**
Sample processing for reading/writing from/to EEPROM

**Declaration**
```c
int32_t Sample_RSPI_EepAccess(int32_t argc, char_t **argv)
```

**Description**
In the sample code, the bit rate is set to 2.78Mbps, and the data of 16KB which has been allocated to the large-capacity on-chip RAM space is written to the EEPROM. Then the data is read from the EEPROM and written to the large-capacity on-chip RAM space.

**Arguments**
- `int32_t argc`: Command arguments input from the terminal
  - Not used in this function.
- `char_t **argv`: Pointer to the command input from the terminal
  - Not used in this function.

**Return Value**
- `COMMAND_SUCCESS`: Success of RSPI sample code processing
- `COMMAND_ERROR`: Failure of RSPI sample code processing

### RSPI_PORT_Init

**Outline**
Initial setting of PORT used in RSPI transmission/reception

**Declaration**
```c
static void RSPI_PORT_Init(void)
```

**Description**
Initializes the PORT used in the RSPI transmission/reception. In the sample code, the PORT is initialized to set the pins of P4_0, P4_1, P4_2, and P4_3 to the functions of RSPCK4, SSL40, MOSI4, and MISO4 respectively.

**Arguments**
None

**Return Value**
None

### Sample_RSPI_EepControlProtect_8bit

**Outline**
Sample processing of EEPROM protection setting or protection release

**Declaration**
```c
void Sample_RSPI_EepControlProtect_8bit(uint32_t channel, uint32_t req)
```

**Description**
Sets or releases the protection to/from the EEPROM using the argument req. In the sample code, the protection for the entire EEPROM area is set or released by writing "B'11" or "B'00" to the BP0 and BP1 bits of the EEPROM status register. The RSPI transfer is executed using the 8-bit data length.

**Arguments**
- `uint32_t channel`: RSPI channel (0 to 4)
- `uint32_t req`: Request for protection release
  - RSPI_EEP_REQ_PROTECT: Request for protection
  - RSPI_EEP_REQ_UNPROTECT: Request for protection release

**Return Value**
None
### Sample_RSPI_EepWrite_8bit

<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>Sample processing for writing to EEPROM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td><code>void Sample_RSPI_EepWrite_8bit(uint32_t channel, uint32_t addr, uint8_t * buffer, uint32_t byte)</code></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Writes the memory contents specified by the argument buffer to the EEPROM for the byte count specified by the argument byte using the memory address specified by the argument addr.</td>
</tr>
<tr>
<td></td>
<td>In the sample code, the sample function RSPI_EepWriteEnable_8bit is called to enable the write operation. This function transmits the data with the EEPROM page write function using the EEPROM instruction code &quot;WRITE(H'02)&quot;, and waits until the write operation is completed.</td>
</tr>
<tr>
<td></td>
<td>The values should be specified for the arguments addr and byte to make the area specified by the said arguments to have the same range as the specified page in the EEPROM.</td>
</tr>
<tr>
<td></td>
<td>The RSPI transfer is executed using the 8-bit data length.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
</tr>
<tr>
<td><code>uint32_t channel</code></td>
<td>RSPI channel (0 to 4)</td>
</tr>
<tr>
<td><code>uint32_t addr</code></td>
<td>Memory address</td>
</tr>
<tr>
<td><code>uint8_t * buffer</code></td>
<td>Store buffer for the data written to the EEPROM</td>
</tr>
<tr>
<td><code>uint32_t byte</code></td>
<td>Byte count to be written to the EEPROM</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### Sample_RSPI_EepRead_8bit

<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>Sample processing for reading from EEPROM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td><code>void Sample_RSPI_EepRead_8bit(uint32_t channel, uint32_t addr, uint8_t * buffer, uint32_t byte)</code></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Reads the data from the EEPROM for the byte count specified by the argument byte using the memory address specified by the argument addr, and stores it to the area specified by the argument buffer.</td>
</tr>
<tr>
<td></td>
<td>In the sample code, the data is read by using the EEPROM instruction code &quot;READ(H'03)&quot;.</td>
</tr>
<tr>
<td></td>
<td>The RSPI transfer is executed using the 8-bit data length.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
</tr>
<tr>
<td><code>uint32_t channel</code></td>
<td>RSPI channel (0 to 4)</td>
</tr>
<tr>
<td><code>uint32_t addr</code></td>
<td>Memory address</td>
</tr>
<tr>
<td><code>uint8_t * buffer</code></td>
<td>Store buffer for the data to be read from the EEPROM</td>
</tr>
<tr>
<td><code>uint32_t byte</code></td>
<td>Byte count to be read from the EEPROM</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
Sample_RSPI_Spti4_Interrupt

Outline
RSPI channel 4 transmit buffer empty interrupt processing

Declaration
void Sample_RSPI_Spti4_Interrupt(uint32_t int_sense)

Description
This is the interrupt handler that runs when the RSPI channel 4 transmit buffer empty interrupt is accepted. It performs the processing to notify that the RSPI channel 4 transmit buffer empty interrupt has been generated.
In the sample code, this function is registered as the INTC handler by using the user-defined function Userdef_RSPI4_Init.
This function calls the API function R_RSPI_SptiInterrupt.

Arguments
uint32_t int_sense : Interrupt detection method (not used)
INTC_LEVEL_SENSITIVE : Level sense
INTC_EDGE_TRIGGER : Edge trigger

Return Value
None

Sample_RSPI_Spri4_Interrupt

Outline
RSPI channel 4 receive buffer full interrupt processing

Declaration
void Sample_RSPI_Spri4_Interrupt(uint32_t int_sense)

Description
This is the interrupt handler that runs when the RSPI channel 4 receive buffer full interrupt is accepted. It performs the processing to notify that the RSPI channel 4 receive buffer full interrupt has been generated.
In the sample code, this function is registered as the INTC handler by using the user-defined function Userdef_RSPI4_Init.
This function calls the API function R_RSPI_SpriInterrupt.

Arguments
uint32_t int_sense : Interrupt detection method (not used)
INTC_LEVEL_SENSITIVE : Level sense
INTC_EDGE_TRIGGER : Edge trigger

Return Value
None
### RSPI_EepWriteStatus_8bit

**Outline**
EEPROM status write processing

**Declaration**
static void RSPI_EepWriteStatus_8bit(uint32_t channel, uint8_t status)

**Description**
Writes the value specified by the argument status to the EEPROM status register.
In the sample code, the sample function RSPI_EepWriteEnable_8bit is called to enable the write operation. Writes the status by using the EEPROM instruction code "WRSR(H'01)" and waits until the write operation is completed.
The RSPI transfer is executed using the 8-bit data length.

**Arguments**
- uint32_t channel
- uint8_t status

**Return Value**
None

### RSPI_EepReadStatus_8bit

**Outline**
EEPROM status read processing

**Declaration**
static uint8_t RSPI_EepReadStatus_8bit(uint32_t channel)

**Description**
Reads the EEPROM status register.
In the sample code, the status is read by using the EEPROM instruction code "RDSR(H'05)". Obtains the read value as a return value.
The RSPI transfer is executed using the 8-bit data length.

**Arguments**
- uint32_t channel

**Return Value**
uint8_t buffer

### RSPI_EepWriteEnable_8bit

**Outline**
EEPROM write-enable processing

**Declaration**
static void RSPI_EepWriteEnable_8bit(uint32_t channel)

**Description**
Enables EEPROM write operation.
In the sample code, the write operation is enabled by using the EEPROM instruction code "WREN(H'06)" to set the "WEL" bit. The RSPI transfer is executed using the 8-bit data length.

**Arguments**
- uint32_t channel

**Return Value**
None

### RSPI_EepBusyWait_8bit

**Outline**
Wait processing for completion of EEPROM write operation

**Declaration**
static void RSPI_EepBusyWait_8bit(uint32_t channel)

**Description**
Reads the EEPROM status register and waits until the "WIP" bit is cleared to "0". In the sample code, the RSPI transfer is executed using the 8-bit data length.

**Arguments**
- uint32_t channel

**Return Value**
None
### RSPI_EepWrite

**Outline**
Processing for data transmission to EEPROM

**Declaration**
static void RSPI_EepWrite(uint32_t channel, void * ope_data, uint32_t ope_count, void * w_data, uint32_t w_data_count)

**Description**
Resets the RSPI transmit and receive buffers to enable the RSPI function. Transmits the instruction code specified by the argument ope_data for the counts specified by the argument ope_count. After the instruction code has been transmitted, the transmit data specified by the argument w_data is transmitted for the counts specified by the argument w_data_count. Waits until the RSPI transmission is completed, and negates the SSL signal by disabling the RSPI function.

The RSPI channel 4 is used in the sample code.

**Arguments**
- `uint32_t channel`: RSPI channel (0 to 4)
- `void * ope_data`: Instruction code (stores instruction code and memory address)
- `uint32_t ope_count`: Transmission counts of instruction code
- `void * w_data`: Transmit data
- `uint32_t w_data_count`: Transmission counts of data

**Return Value**
None

**Notes**
The transmission methods differ depending on the "instruction code" and "transmit data" specified by the arguments. In this sample function, the API function R_RSPI_Write is used in the data transmission. The R_RSPI_Write function waits for satisfaction of the transmit buffer empty conditions prior to the data transmission and stores the data in the RSPI transmit buffer. The transmit buffer empty conditions are satisfied based on the value of the TXTRG bits in the buffer control register (SPBFCR).

In the transmission of the "instruction code", the data is transmitted after the value of the TXTRG bits is changed to "0"(1 byte empty space in SPTX), therefore the data is stored in the transmit buffer as soon as 1 or more bytes in the transmit buffer become empty.

In the transmission of the "transmit data", the setting value of the TXTRG bits is not changed. The TXTRG is set to "3" by using the API function R_RSPI_Init (8 bytes empty space in SPTX) in the sample code. The data up to 8 bytes is stored in the transmit buffer after waiting until all the data space in the transmit buffer become empty.
RSPI_EepRead

Outline
Processing for data reception from EEPROM

Declaration
static void RSPI_EepRead(uint32_t channel, void * ope_data, uint32_t ope_count,
void * r_data, uint32_t r_data_count, uint32_t r_byte_flg)

Description
Receives the data from the EEPROM using the RSPI specified by the argument channel.
Resets the RSPI transmit and receive buffers to enable the RSPI function. Transmits
the instruction code specified by the argument ope_data for the counts specified by
the argument ope_count. Then resets the RSPI buffer and receives the data for the
counts specified by the argument r_data_count, which is stored to the area specified
by the argument r_data. Waits until the RSPI transmission is completed, and negates
the SSL signal by disabling the RSPI function. If the receive data is smaller than the
byte count specified for the RXTRG bits such as the EEPROM status register (1 byte) is read, EEP_0BYTE_READ should be set to r_byte_flg.

Arguments
uint32_t channel : RSPI channel (0 to 4)
void * ope_data : Instruction code
(void stores instruction code and memory address)
uint32_t ope_count : Transmission counts of instruction code
void * r_data : Receive data
uint32_t r_data_count : Reception counts of data
uint32_t r_byte_flg : Select flag for received byte count
EEP_nBYTE_READ : When receiving normal data
EEP_0BYTE_READ : When receiving data smaller than the bytes specified by the RXTRG bits

Return Value
None

Notes
In this sample function, the API function R_RSPI_Read is used to receive the data to
be stored in the argument r_data. The R_RSPI_Read function waits for satisfaction
of the receive buffer full conditions prior to the data reception and obtains the data
from the RSPI receive buffer. The receive buffer full conditions are satisfied based on
the value of the RXTRG bits in the buffer control register (SPBFCR). The
R_RSPI_Read function does not change the setting value of the RXTRG bits.
In the sample code, the RXTRG is set to "5" by using the API function R_RSPI_Init,
therefore the data is obtained from the receive buffer after waiting until 24 bytes of
data is stored in the receive buffer. However, when EEP_0BYTE_READ is specified
for the argument r_byte_flg, the R_RSPI_Read function obtains the data from the
receive buffer as soon as 1 or more bytes of data is stored because the
R_RSPI_Read function receives the data after setting "0" to the RXTRG bits.
An efficient data reception can be executed by specifying EEP_0BYTE_READ for
the argument r_byte_flg within the processing of tasks such as the reception of the
EEPROM status register value (1 byte).
### R_RSPI_Init

**Outline**
RSPI initial setting

**Declaration**

```c
int32_t R_RSPI_Init(uint32_t channel)
```

**Description**
Initializes the RSPI specified by the argument channel by using the user-defined function Userdef_RSPIn_Init (n=0 to 4).

**Arguments**
- `uint32_t channel`: RSPI channel (0 to 4)

**Return Value**
- `DEVDRV_SUCCESS`: Success of RSPI initialization
- `DEVDRV_ERROR`: Failure of RSPI initialization

### R_RSPI_Open

**Outline**
RSPI function enable setting

**Declaration**

```c
int32_t R_RSPI_Open(uint32_t channel)
```

**Description**
Enables the RSPI function specified by the argument channel.

**Arguments**
- `uint32_t channel`: RSPI channel (0 to 4)

**Return Value**
- `DEVDRV_SUCCESS`: Success in starting RSPI communication
- `DEVDRV_ERROR`: Failure in starting RSPI communication

### R_RSPI_Close

**Outline**
RSPI function disable setting

**Declaration**

```c
int32_t R_RSPI_Close(uint32_t channel)
```

**Description**
Disables the RSPI function specified by the argument channel, and negates the SSL signal.

When using the device which is required to negate the SSL signal before the next access starts, the processing to secure the SSL signal negation period should be executed by using the user-defined function Userdef_RSPIn_DelayNextAccess (n=0 to 4).

**Arguments**
- `uint32_t channel`: RSPI channel (0 to 4)

**Return Value**
- `DEVDRV_SUCCESS`: Success in ending RSPI communication
- `DEVDRV_ERROR`: Failure in ending RSPI communication
### R_RSPI_Write

| **Outline** | RSPI transmit processing |
| **Declaration** | int32_t R_RSPI_Write(uint32_t channel, void * buffer, uint32_t count, uint8_t txtrg) |
| **Description** | Transmits data on the RSPI channel by the argument channel. The empty space for the byte count set to the TXTRG bits in the buffer control register (SPBFCR) can be secured in the RSPI transmit buffer by waiting for the satisfaction of the conditions for the transmit buffer empty notification information using the user-defined function Userdef_RSPIn_WaitTransmitEmpty. Writes the data to empty space in the transmit buffer until the space becomes full and initializes the transmit buffer empty notification information by using the user-defined function Userdef_RSPIn_InitTransmitEmpty (n=0 to 4). The transmit operation continues until the transmission count reaches the number specified by the argument count. At the start of this function, the TXTRG bits are set to the value specified by the argument txtrg, and the original value is returned at the end of the function. However, the TXTRG bits are not changed if RSPI_TXTRG_CURRENT is specified for the argument txtrg. The access width specified by the SPLW bits in the data control register (SPDCR) is used for data transmissions. Dummy read is performed each time a transmission occurs when the receive buffer is full to prevent a receive overflow. |

| **Arguments** | 
| uint32_t channel : RSPI channel (0 to 4) |
| void * buffer : Transmit data |
| uint32_t count : Transmission counts of data |
| uint8_t txtrg : Transmit buffer data triggering number (Setting values for TXTRG bits) |
| RSPI_TXTRG_CURRENT : Current setting value. |
| RSPI_TXTRG_07_BYTE : Change to 7 bytes |
| RSPI_TXTRG_06_BYTE : Change to 6 bytes |
| RSPI_TXTRG_04_BYTE : Change to 4 bytes |
| RSPI_TXTRG_00_BYTE : Change to 0 byte |

| **Return Value** | 
| DEVDRV_SUCCESS : Success of RSPI transmission |
| DEVDRV_ERROR : Failure of RSPI transmission |
### R_RSPI_Read

**Outline**
RSPI receive processing

**Declaration**
```c
int32_t R_RSPI_Read(uint32_t channel, void * buffer, uint32_t count, uint8_t rxtrg)
```

**Description**
Receives data using the RSPI channel specified by the argument channel. The data for the byte count set to the RXTRG bits in the buffer control register (SPBFCR) is stored in the receive buffer by waiting for the satisfaction of the conditions for the receive buffer full notification information using the user-defined function Userdef_RSPIn_WaitReceiveFull. Receives the data for the byte count and initializes the receive buffer full notification information by using the user-defined function Userdef_RSPIn_InitReceiveFull (n=0 to 4). The receive operation continues until reception count reaches to the number specified by the argument count. If the reception count specified by the argument count is less than the byte count set to the RXTRG bits, only the data specified by the argument count is received.

At the start of this function, the RXTRG bits are set to the value specified by the argument rxtrg, and RXTRG bits are returned to the original values at the end of the function. However, the RXTRG bits are not changed if RSPI_RXTRG_CURRENT is specified for the argument rxtrg. The access width specified by the SPLW bits in the data control register (SPDCR) is used for data reception.

**Arguments**
- `uint32_t channel`: RSPI channel (0 to 4)
- `void * buffer`: Receive data
- `uint32_t count`: Reception counts of data
- `uint8_t rxtrg`: Receive buffer data triggering number (Setting values for RXTRG bits)
  - RSPI_RXTRG_CURRENT: Current setting value.
  - RSPI_RXTRG_01_BYTE: Change to 1 byte.
  - RSPI_RXTRG_02_BYTE: Change to 2 bytes.
  - RSPI_RXTRG_04_BYTE: Change to 4 bytes.
  - RSPI_RXTRG_08_BYTE: Change to 8 bytes.
  - RSPI_RXTRG_16_BYTE: Change to 16 bytes.
  - RSPI_RXTRG_24_BYTE: Change to 24 bytes.
  - RSPI_RXTRG_32_BYTE: Change to 32 bytes.
  - RSPI_RXTRG_05_BYTE: Change to 5 bytes.

**Return Value**
- `DEVDRV_SUCCESS`: Success of RSPI reception
- `DEVDRV_ERROR`: Failure of RSPI reception

### R_RSPI_ResetBuffer

**Outline**
RSPI buffer reset processing

**Declaration**
```c
int32_t R_RSPI_ResetBuffer(uint32_t channel)
```

**Description**
Resets the transmit buffer and receive buffers of the RSPI specified by the argument channel.

Initializes the transmit buffer empty notification information and the receive buffer full notification information by using the user-defined functions Userdef_RSPIn_InitTransmitEmpty and Userdef_RSPIn_InitTransmitEmpty (n=0 to 4).

**Arguments**
- `uint32_t channel`: RSPI channel (0 to 4)

**Return Value**
- `DEVDRV_SUCCESS`: Success of RSPI buffer reset
- `DEVDRV_ERROR`: Failure of RSPI buffer reset
### R_RSPI_WaitTransmitEnd

**Outline**
Wait processing for RSPI end of transmission

**Declaration**
```c
int32_t R_RSPI_WaitTransmitEnd(uint32_t channel)
```

**Description**
Waits for transmit end on the RSPI channel specified by the argument channel.
Waits until the conditions for the end of transmission are satisfied by using the user-defined function `Userdef_RSPIn_WaitTransmitEnd (n=0 to 4)`.

**Arguments**
- `uint32_t channel`: RSPI channel (0 to 4)

**Return Value**
- `DEVDRV_SUCCESS`: Success of RSPI transmit end waiting
- `DEVDRV_ERROR`: Failure of RSPI transmit end waiting

### R_RSPI_SptiInterrupt

**Outline**
RSPI transmit buffer empty interrupt notification processing

**Declaration**
```c
int32_t R_RSPI_SptiInterrupt(uint32_t channel)
```

**Description**
Executes the processing to satisfy the conditions for the transmit buffer empty notification information of the RAPI specified by the argument channel.
This function is called by the interrupt handler of the RSPI transmit buffer empty interrupt to notify that an interrupt has been generated.
The processing to satisfy the conditions for the transmit buffer empty notification information should be executed by using the user-defined function `Userdef_RSPIn_SetTransmitEmpty (n=0 to 4)`.

**Arguments**
- `uint32_t channel`: RSPI channel (0 to 4)

**Return Value**
- `DEVDRV_SUCCESS`: Success of RSPI transmit buffer empty interrupt notification
- `DEVDRV_ERROR`: Failure of RSPI transmit buffer empty interrupt notification

### R_RSPI_SpriInterrupt

**Outline**
RSPI receive buffer full interrupt notification processing

**Declaration**
```c
int32_t R_RSPI_SpriInterrupt(uint32_t channel)
```

**Description**
Executes the processing to satisfy the conditions for the receive buffer full notification information of the RSPI specified by the argument channel.
This function is called by the interrupt handler of the RSPI receive buffer full interrupt to notify that an interrupt has been generated. The processing to satisfy the conditions or the receive buffer full notification information should be executed by using the user-defined function `Userdef_RSPIn_SetReceiveFull (n=0 to 4)`.

**Arguments**
- `uint32_t channel`: RSPI channel (0 to 4)

**Return Value**
- `DEVDRV_SUCCESS`: Success of RSPI receive buffer full interrupt notification
- `DEVDRV_ERROR`: Failure of RSPI receive buffer full interrupt notification
Userdef_RSPI0_Init

Outline: RSPI channel 0 initial setting
Declaration: void Userdef_RSPI0_Init(void)
Description: This is a user-defined function. The RSPI initial setting and the setting for interrupts are required when the RSPI channel 0 is sued.
In the sample program, this function returns without any processing.
Arguments: None
Return Value: None

Userdef_RSPI1_Init

Outline: RSPI channel 1 initial setting
Declaration: void Userdef_RSPI1_Init(void)
Description: This is a user-defined function. The RSPI initial setting and the setting for interrupts are required when the RSPI channel 1 is sued.
In the sample program, this function returns without any processing.
Arguments: None
Return Value: None

Userdef_RSPI2_Init

Outline: RSPI channel 2 initial setting
Declaration: void Userdef_RSPI2_Init(void)
Description: This is a user-defined function. The RSPI initial setting and the setting for interrupts are required when the RSPI channel 2 is sued.
In the sample program, this function returns without any processing.
Arguments: None
Return Value: None
### Userdef_RSPI3_Init

**Outline**
RSPI channel 3 initial setting

**Declaration**
void Userdef_RSPI3_Init(void)

**Description**
This is a user-defined function. The RSPI initial setting and the setting for interrupts are required when the RSPI channel 3 is sued. In the sample program, this function returns without any processing.

**Arguments**
None

**Return Value**
None

### Userdef_RSPI4_Init

**Outline**
RSPI channel 4 initial setting

**Declaration**
void Userdef_RSPI4_Init(void)

**Description**
This is a user-defined function. Initial setting is required to use the RSPI channel 4. In the sample code, the RSPI transfer is executed in the master mode by setting the bit rage of 2.78Mbps (= P1/f/24), MSB first, and the 8-bit data length. "1" is set to the RSPCK signal as polarity when idle. The data is changed at the falling edge and sample at the rising edge. The SSL signal is set as "0"-active. The RSPI setting is executed to set the transmit buffer empty flag when the data in the transmit buffer has 0 byte or smaller, and the receive buffer full flag when the data in the receive buffer has 24 bytes or more. The INTC setting is also executed to use the transmit buffer empty interrupt and the receive buffer full interrupt. In this function, the RSPI is set to disable the transmission and the reception interrupts.

**Arguments**
None

**Return Value**
None
<table>
<thead>
<tr>
<th><strong>Userdef_RSPI0_InitTransmitEmpty</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
<td>Initialization of RSPI channel 0 transmit buffer empty notification information</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>void Userdef_RSPI0_InitTransmitEmpty(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a user-defined function. Execute this function to initialize the RSPI channel 0 transmit buffer empty notification information when the RSPI channel 0 is used. In the sample code, control is returned with no processing.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Userdef_RSPI1_InitTransmitEmpty</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
<td>Initialization of RSPI channel 1 transmit buffer empty notification information</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>void Userdef_RSPI1_InitTransmitEmpty(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a user-defined function. Execute this function to initialize the RSPI channel 1 transmit buffer empty notification information when the RSPI channel 1 is used. In the sample code, control is returned with no processing.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Userdef_RSPI2_InitTransmitEmpty</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
<td>Initialization of RSPI channel 2 transmit buffer empty notification information</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>void Userdef_RSPI2_InitTransmitEmpty(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a user-defined function. Execute this function to initialize the RSPI channel 2 transmit buffer empty notification information when the RSPI channel 2 is used. In the sample code, control is returned with no processing.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
Userdef_RSPI3_InitTransmitEmpty

Outline  Initialization of RSPI channel 3 transmit buffer empty notification information
Declaration  void Userdef_RSPI3_InitTransmitEmpty(void)
Description  This is a user-defined function.
Execute this function to initialize the RSPI channel 3 transmit buffer empty notification information when the RSPI channel 3 is used.
In the sample code, control is returned with no processing.
Arguments  None
Return Value  None

Userdef_RSPI4_InitTransmitEmpty

Outline  Initialization of RSPI channel 4 transmit buffer empty notification information
Declaration  void Userdef_RSPI4_InitTransmitEmpty(void)
Description  This is a user-defined function.
Execute this function to initialize the RSPI channel 4 transmit buffer empty notification information.
In the sample code, "0" is set to the software flag to notify that the RSPI channel 4 transmit buffer empty interrupt has been generated.
Arguments  None
Return Value  None
### Userdef_RSPI0_InitReceiveFull

**Outline**
Initialization of RSPI channel 0 receive buffer full notification information

**Declaration**
```c
void Userdef_RSPI0_InitReceiveFull(void)
```

**Description**
This is a user-defined function.
Execute this function to initialize the RSPI channel 0 receive buffer full notification information when the RSPI channel 0 is used.
In the sample code, this function returns without any processing.

**Arguments**
None

**Return Value**
None

### Userdef_RSPI1_InitReceiveFull

**Outline**
Initialization of RSPI channel 1 receive buffer full notification information

**Declaration**
```c
void Userdef_RSPI1_InitReceiveFull(void)
```

**Description**
This is a user-defined function.
Execute this function to initialize the RSPI channel 1 receive buffer full notification information when the RSPI channel 1 is used.
In the sample code, this function returns without any processing.

**Arguments**
None

**Return Value**
None

### Userdef_RSPI2_InitReceiveFull

**Outline**
Initialization of RSPI channel 2 receive buffer full notification information

**Declaration**
```c
void Userdef_RSPI2_InitReceiveFull(void)
```

**Description**
This is a user-defined function.
Execute this function to initialize the RSPI channel 2 receive buffer full notification information when the RSPI channel 2 is used.
In the sample code, this function returns without any processing.

**Arguments**
None

**Return Value**
None
Userdef_RSPI3_InitReceiveFull

Outline  Initialization of RSPI channel 3 receive buffer full notification information
Declaration void Userdef_RSPI3_InitReceiveFull(void)
Description This is a user-defined function.
Execute this function to initialize the RSPI channel 3 receive buffer full notification information when the RSPI channel 3 is used.
In the sample code, this function returns without any processing.
Arguments None
Return Value None

Userdef_RSPI4_InitReceiveFull

Outline  Initialization of RSPI channel 4 receive buffer full notification information
Declaration void Userdef_RSPI4_InitReceiveFull(void)
Description This is a user-defined function.
Execute this function to initialize the RSPI channel 4 receive buffer full notification information.
In the sample code, "0" is set to the software flag to notify that the RSPI channel 4 receive buffer full interrupt has been generated.
Arguments None
Return Value None
<table>
<thead>
<tr>
<th>Function: Userdef_RSPI0_SetTransmitEmpty</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
</tr>
<tr>
<td>Setting to satisfy the conditions for RSPI channel 0 transmit buffer empty notification information</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
</tr>
<tr>
<td>void Userdef_RSPI0_SetTransmitEmpty(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This is a user-defined function.</td>
</tr>
<tr>
<td>Execute this function to satisfy the conditions for the RSPI channel 0 transmit buffer empty notification information when the RSPI channel 0 is used.</td>
</tr>
<tr>
<td>In the sample code, this function returns without any processing.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
</tr>
<tr>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
</tr>
<tr>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function: Userdef_RSPI1_SetTransmitEmpty</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
</tr>
<tr>
<td>Setting to satisfy the conditions for RSPI channel 1 transmit buffer empty notification information</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
</tr>
<tr>
<td>void Userdef_RSPI1_SetTransmitEmpty(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This is a user-defined function.</td>
</tr>
<tr>
<td>Execute this function to satisfy the conditions for the RSPI channel 1 transmit buffer empty notification information when the RSPI channel 1 is used.</td>
</tr>
<tr>
<td>In the sample code, this function returns without any processing.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
</tr>
<tr>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
</tr>
<tr>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function: Userdef_RSPI2_SetTransmitEmpty</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
</tr>
<tr>
<td>Setting to satisfy the conditions for RSPI channel 2 transmit buffer empty notification information</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
</tr>
<tr>
<td>void Userdef_RSPI2_SetTransmitEmpty(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This is a user-defined function.</td>
</tr>
<tr>
<td>Execute this function to satisfy the conditions for the RSPI channel 2 transmit buffer empty notification information when the RSPI channel 2 is used.</td>
</tr>
<tr>
<td>In the sample code, this function returns without any processing.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
</tr>
<tr>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
</tr>
<tr>
<td>None</td>
</tr>
</tbody>
</table>
### Userdef_RSPI3_SetTransmitEmpty

**Outline**
Setting to satisfy the conditions for RSPI channel 3 transmit buffer empty notification information

**Declaration**
void Userdef_RSPI3_SetTransmitEmpty(void)

**Description**
This is a user-defined function.
Execute this function to satisfy the conditions for the RSPI channel 3 transmit buffer empty notification information when the RSPI channel 3 is used.
In the sample code, this function returns without any processing.

**Arguments**
None

**Return Value**
None

### Userdef_RSPI4_SetTransmitEmpty

**Outline**
Setting to satisfy the conditions for RSPI channel 4 transmit buffer empty notification information

**Declaration**
void Userdef_RSPI4_SetTransmitEmpty(void)

**Description**
This is a user-defined function.
Execute this function to satisfy the conditions for the RSPI channel 4 transmit buffer empty notification information.
In the sample code, "1" is set to the software flag to notify that the RSPI channel 4 transmit buffer empty interrupt has been generated.
This function disables the transmit buffer empty interrupt.

**Arguments**
None

**Return Value**
None
### Userdef_RSPI0_SetReceiveFull

<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>Setting to satisfy the conditions for RSPI channel 0 receive buffer full notification information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td>void Userdef_RSPI0_SetReceiveFull(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a user-defined function. Execute this function to satisfy the conditions for the RSPI channel 0 receive buffer full notification information when the RSPI channel 0 is used. In the sample code, this function returns without any processing.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### Userdef_RSPI1_SetReceiveFull

<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>Setting to satisfy the conditions for RSPI channel 1 receive buffer full notification information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td>void Userdef_RSPI1_SetReceiveFull(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a user-defined function. Execute this function to satisfy the conditions for the RSPI channel 1 receive buffer full notification information when the RSPI channel 1 is used. In the sample code, this function returns without any processing.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### Userdef_RSPI2_SetReceiveFull

<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>Setting to satisfy the conditions for RSPI channel 2 receive buffer full notification information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td>void Userdef_RSPI2_SetReceiveFull(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a user-defined function. Execute this function to satisfy the conditions for the RSPI channel 2 receive buffer full notification information when the RSPI channel 2 is used. In the sample code, this function returns without any processing.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
### Userdef_RSPI3_SetReceiveFull

<table>
<thead>
<tr>
<th>Outline</th>
<th>Setting to satisfy the conditions for RSPI channel 3 receive buffer full notification information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>void Userdef_RSPI3_SetReceiveFull(void)</td>
</tr>
</tbody>
</table>
| Description | This is a user-defined function.  
Execute this function to satisfy the conditions for the RSPI channel 3 receive buffer full notification information when the RSPI channel 3 is used.  
In the sample code, this function returns without any processing. |
| Arguments | None |
| Return Value | None |

### Userdef_RSPI4_SetReceiveFull

<table>
<thead>
<tr>
<th>Outline</th>
<th>Setting to satisfy the conditions for RSPI channel 4 receive buffer full notification information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>void Userdef_RSPI4_SetReceiveFull(void)</td>
</tr>
</tbody>
</table>
| Description | This is a user-defined function.  
Execute this function to satisfy the conditions for the RSPI channel 4 receive buffer full notification information.  
In the sample code, "1" is set to the software flag to notify that the RSPI channel 4 receive buffer full interrupt has been generated.  
This function disables the receive buffer full interrupt. |
| Arguments | None |
| Return Value | None |
### Userdef_RSPI0_WaitTransmitEmpty

**Outline**
Waiting for satisfaction of the conditions for RSPI channel 0 transmit buffer empty notification information

**Declaration**
void Userdef_RSPI0_WaitTransmitEmpty(void)

**Description**
This is a user-defined function.
Execute this function to wait until the conditions for the RSPI channel 0 transmit buffer empty notification information are satisfied when the RSPI channel 0 is used.
In the sample code, this function returns without any processing.

**Arguments**
None

**Return Value**
None

---

### Userdef_RSPI1_WaitTransmitEmpty

**Outline**
Waiting for satisfaction of the conditions for RSPI channel 1 transmit buffer empty notification information

**Declaration**
void Userdef_RSPI1_WaitTransmitEmpty(void)

**Description**
This is a user-defined function.
Execute this function to wait until the conditions for the RSPI channel 1 transmit buffer empty notification information are satisfied when the RSPI channel 1 is used.
In the sample code, this function returns without any processing.

**Arguments**
None

**Return Value**
None

---

### Userdef_RSPI2_WaitTransmitEmpty

**Outline**
Waiting for satisfaction of the conditions for RSPI channel 2 transmit buffer empty notification information

**Declaration**
void Userdef_RSPI2_WaitTransmitEmpty(void)

**Description**
This is a user-defined function.
Execute this function to wait until the conditions for the RSPI channel 2 transmit buffer empty notification information are satisfied when the RSPI channel 2 is used.
In the sample code, this function returns without any processing.

**Arguments**
None

**Return Value**
None
Userdef_RSPI3_WaitTransmitEmpty

Outline
Waiting for satisfaction of the conditions for RSPI channel 3 transmit buffer empty notification information

Declaration
void Userdef_RSPI3_WaitTransmitEmpty(void)

Description
This is a user-defined function.
Execute this function to wait until the conditions for the RSPI channel 3 transmit buffer empty notification information are satisfied when the RSPI channel 3 is used.
In the sample code, this function returns without any processing.

Arguments
None

Return Value
None

Userdef_RSPI4_WaitTransmitEmpty

Outline
Waiting for satisfaction of the conditions for RSPI channel 4 transmit buffer empty notification information

Declaration
void Userdef_RSPI4_WaitTransmitEmpty(void)

Description
This is a user-defined function.
Execute this function to wait until the conditions for the RSPI channel 4 transmit buffer empty notification information are satisfied.
In the sample code, this function enables the transmit buffer empty interrupt and waits until "1" is set to the software flag to notify that the interrupt has been generated.

Arguments
None

Return Value
None
### Userdef_RSPI0_WaitTransmitEnd

**Outline**
Waiting for RSPI channel 0 end of transmission

**Declaration**
void Userdef_RSPI0_WaitTransmitEnd(void)

**Description**
This is a user-defined function.
Execute this function to wait until the conditions for the RSPI channel 0 end of transmission are satisfied when the RSPI channel 0 is used.
In the sample code, this function returns without any processing.

**Arguments**
None

**Return Value**
None

### Userdef_RSPI1_WaitTransmitEnd

**Outline**
Waiting for RSPI channel 1 end of transmission

**Declaration**
void Userdef_RSPI1_WaitTransmitEnd(void)

**Description**
This is a user-defined function.
Execute this function to wait until the conditions for the RSPI channel 1 end of transmission are satisfied when the RSPI channel 1 is used.
In the sample code, this function returns without any processing.

**Arguments**
None

**Return Value**
None

### Userdef_RSPI2_WaitTransmitEnd

**Outline**
Waiting for RSPI channel 2 end of transmission

**Declaration**
void Userdef_RSPI2_WaitTransmitEnd(void)

**Description**
This is a user-defined function.
Execute this function to wait until the conditions for the RSPI channel 2 end of transmission are satisfied when the RSPI channel 2 is used.
In the sample code, this function returns without any processing.

**Arguments**
None

**Return Value**
None
<table>
<thead>
<tr>
<th>Userdef_RSPI3_WaitTransmitEnd</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Userdef_RSPI4_WaitTransmitEnd</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
</tr>
</tbody>
</table>
Userdef_RSPI0_WaitReceiveFull

Outline  Waiting for satisfaction of the conditions for RSPI channel 0 receive buffer full notification information

Declaration  void Userdef_RSPI0_WaitReceiveFull(uint32_t byte, uint32_t less_rxtrg)

Description  This is a user-defined function.
Execute this function to wait until the conditions for the RSPI channel 0 receive buffer full notification information are satisfied when the RSPI channel 0 is used.
In the sample code, this function returns without any processing.

Arguments  
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t byte</td>
<td>Byte count of dummy data to be transmitted</td>
</tr>
<tr>
<td>uint32_t less_rxtrg</td>
<td>Setting for operation</td>
</tr>
</tbody>
</table>

  0: Normal operation
  (Wait until the conditions for the receive buffer full notification information are satisfied.)
  1: Wait until the conditions for the receive buffer full notification information are satisfied, but the conditions are determined they have been satisfied when the data is received for the bytes specified by the argument byte.

Return Value  None

Userdef_RSPI1_WaitReceiveFull

Outline  Waiting for satisfaction of the conditions for RSPI channel 1 receive buffer full notification information

Declaration  void Userdef_RSPI1_WaitReceiveFull(uint32_t byte, uint32_t less_rxtrg)

Description  This is a user-defined function.
Execute this function to wait until the conditions for the RSPI channel 1 receive buffer full notification information are satisfied when the RSPI channel 1 is used.
In the sample code, this function returns without any processing.

Arguments  
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t byte</td>
<td>Byte count of dummy data to be transmitted</td>
</tr>
<tr>
<td>uint32_t less_rxtrg</td>
<td>Setting for operation</td>
</tr>
</tbody>
</table>

  0: Normal operation
  (Wait until the conditions for the receive buffer full notification information are satisfied.)
  1: Wait until the conditions for the receive buffer full notification information are satisfied, but the conditions are determined they have been satisfied when the data is received for the bytes specified by the argument byte.

Return Value  None
### Userdef_RSPI2_WaitReceiveFull

<table>
<thead>
<tr>
<th>Outline</th>
<th>Waiting for satisfaction of the conditions for RSPI channel 2 receive buffer full notification information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td><code>void Userdef_RSPI2_WaitReceiveFull(uint32_t byte, uint32_t less_rxtrg)</code></td>
</tr>
<tr>
<td>Description</td>
<td>This is a user-defined function. Use this function to wait until the conditions for the RSPI channel 2 receive buffer full notification information are satisfied when the RSPI channel 2 is used. In the sample code, this function returns without any processing.</td>
</tr>
</tbody>
</table>
| Arguments | `uint32_t byte`: Byte count of dummy data to be transmitted  
               `uint32_t less_rxtrg`: Setting for operation  
               0: Normal operation  
               (Wait until the conditions for the receive buffer full notification information are satisfied.)  
               1: Wait until the conditions for the receive buffer full notification information are satisfied, but the conditions are determined they have been satisfied when the data is received for the bytes specified by the argument byte. |
| Return Value | None |

### Userdef_RSPI3_WaitReceiveFull

<table>
<thead>
<tr>
<th>Outline</th>
<th>Waiting for satisfaction of the conditions for RSPI channel 3 receive buffer full notification information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td><code>void Userdef_RSPI3_WaitReceiveFull(uint32_t byte, uint32_t less_rxtrg)</code></td>
</tr>
<tr>
<td>Description</td>
<td>This is a user-defined function. Use this function to wait until the conditions for the RSPI channel 3 receive buffer full notification information are satisfied when the RSPI channel 3 is used. In the sample code, this function returns without any processing.</td>
</tr>
</tbody>
</table>
| Arguments | `uint32_t byte`: Byte count of dummy data to be transmitted  
               `uint32_t less_rxtrg`: Setting for operation  
               0: Normal operation  
               (Wait until the conditions for the receive buffer full notification information are satisfied.)  
               1: Wait until the conditions for the receive buffer full notification information are satisfied, but the conditions are determined they have been satisfied when the data is received for the bytes specified by the argument byte. |
| Return Value | None |
### Userdef_RSPI4_WaitReceiveFull

#### Outline
Waiting for satisfaction of the conditions for RSPI channel 4 receive buffer full notification information

#### Declaration
```c
void Userdef_RSPI4_WaitReceiveFull(uint32_t byte, uint32_t less_rxtrg);
```

#### Description
This is a user-defined function.
Execute this function to wait until the conditions for the RSPI channel 4 receive buffer full notification information are satisfied.
In the sample code, the dummy data is transmitted to output the RSPCK clock in order to receive the data for the bytes specified by the argument byte.
Enables the RSPI channel 4 receive buffer full interrupt and waits until "1" is set to the software flag to notify that the interrupt has been generated. However, if "1" is specified for the argument less_rxtrg, terminate this function after satisfying the conditions for the receive buffer full notification information when the data is stored in the receive buffer for the bytes specified by the argument byte.
When receiving the byte count which is smaller than the one specified by the RXTRG bits in the buffer control register (SPBFCR), efficient receive processing can be executed by setting "1" to the argument less_rxtrg.

#### Arguments
- `uint32_t byte`: Byte count of dummy data to be transmitted
- `uint32_t less_rxtrg`: Setting for operation
  - 0: Normal operation
    (Wait until the conditions for the receive buffer full notification information are satisfied.)
  - 1: Wait until the conditions for the receive buffer full notification information are satisfied, but the conditions are determined they have been satisfied when the data is received for the bytes specified by the argument byte.

#### Return Value
None
### Userdef_RSPI0_DelayNextAccess

<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>Securing period to start next access on RSPI channel 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td>void Userdef_RSPI0_DelayNextAccess(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a user-defined function.</td>
</tr>
<tr>
<td></td>
<td>When the RSPI channel 0 is used, execute this function to secure the period which is required to start the next access after the transfer ends.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### Userdef_RSPI1_DelayNextAccess

<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>Securing period to start next access on RSPI channel 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td>void Userdef_RSPI1_DelayNextAccess(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a user-defined function.</td>
</tr>
<tr>
<td></td>
<td>When the RSPI channel 1 is used, execute this function to secure the period which is required to start the next access after the transfer ends.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### Userdef_RSPI2_DelayNextAccess

<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>Securing period to start next access on RSPI channel 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td>void Userdef_RSPI2_DelayNextAccess(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a user-defined function.</td>
</tr>
<tr>
<td></td>
<td>When the RSPI channel 2 is used, execute this function to secure the period which is required to start the next access after the transfer ends.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
<tr>
<td>Userdef_RSPI3_DelayNextAccess</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Outline</strong></td>
<td>Securing period to start next access on RSPI channel 3</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>void Userdef_RSPI3_DelayNextAccess(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This is a user-defined function. When the RSPI channel 3 is used, execute this function to secure the period which is required to start the next access after the transfer ends. In the sample code, this function returns without any processing.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Userdef_RSPI4_DelayNextAccess</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
</tr>
</tbody>
</table>
5.9 Flowcharts

5.9.1 Main Processing

Figure 5.7 shows the Main Processing.

- main
  - Output to terminal printf()
    - Outputs the sample code version information to the terminal software running on the host PC connected via the serial interface.
  - OSTM0-related settings
    - Blinks the LED at 500ms intervals using the OSTM channel 0 interrupt. Refer to the RZ/A1H group Example of Initialization application note for details.
  - Peripheral function sample code startup function Sample_Main()
    - Branches to processing that waits to receive a command from the terminal. The sample code for the appropriate peripheral function is executed according to the command received.
  - return (0)
5.9.2 Sample Code Main Processing

Figure 5.8 shows the flowchart of Sample Code Main Function. This function waits for the character input from the terminal software running on the host PC.

The RSPI sample code is executed when "RSPI" + "Enter" key is input.

```
Sample_Main

Output to terminal
print()

Acquisition of main
processing command list
Sample_GetCmdList()

Registration of main
processing command list
CommandSetCmdList()

Wait for command input
gets()

Analyze and execute
command
CommandExe()

*EXIT* input?

*EXIT* was input

Reacquisition of main
processing command list
Sample_GetCmdList()

*EXIT* was not input

Reregistration of main
processing command list
CommandSetCmdList()
```

Figure 5.8 Sample Code Main Function
5.9.3 RSPI Sample Main Function

Figure 5.9 shows the flowchart of RSPI Sample Main Function. This function waits for character input from the terminal software running on the host PC and branches to the RSPI sample code according to the input command.

When "1" + "Enter" key is input, the sample code is executed, which writes/read the 16KB data to/from the EEPROM at the bit rate of 2.78 Mbps.

```
Sample_RSPI_Main

Output to terminal
printf()

Acquisition of RSPI sample
processing command list
Sample_RSPI_GetCmdList()

Registration of main
processing command list
CommandSetCmdList()

Wait for command input
gets()

Analyze and execute
command
CommandExe()

(EXIT) input?

(EXIT) was not input

(EXIT) was input

return (COMMAND_EXIT)
```

Outputs the RSPI sample program version information to the terminal running on the host PC.

Obtains the RSPI sample processing command list. The menu list for launching the RSPI sample code is obtained in this sample code.

Registers the RSPI sample processing command list. The menu list for launching the RSPI sample code is registered in this sample code.

Waits for command input from the terminal and stores it in the command buffer.

Analyzes and executes the contents of the command buffer. In the sample code, this function branches to one of the following RSPI sample code processing routines, according to the input command:

- "1" : Branches to the Sample_RSPI_EepAccess function.
- "HELP" : Displays a list of the available commands.
- "EXIT" input?
  - "EXIT" was input
    - return (COMMAND_EXIT)
  - "EXIT" was not input
    - Analyze and execute command
    - CommandExe()
```

Figure 5.9   RSPI Sample Main Function
5.9.4 Sample Function for Reading/Writing From/To EEPROM

Figure 5.10 shows the flowchart of Sample Function for Reading/Writing From/To EEPROM.

- **Sample_RSPI_EepAccess**
  - Transmit buffer initialization
  - Receive buffer initialization
  - PORT initial settings
    - RSPI_PORT_Init()
  - RSPI initial settings
    - R_RSPI_Init()
  - EEPROM protection release
    - Sample_RSPI_EepControlProtect_8bit()
  - Transmit data to EEPROM
    - Sample_RSPI_EepWrite_8bit()
  - Receive data from EEPROM
    - Sample_RSPI_EepRead_8bit()
  - EEPROM protection
    - Sample_RSPI_EepControlProtect_8bit()

- **return (COMMAND_SUCCESS)**

Generates data for transmission to the EEPROM (data incremented from "0" to "255") and stores 16KB of it in the transmit buffer (trans_data[]).

Initializes the store buffer for receive data (receive_data[]) from the EEPROM by "0".

Initializes the PORT used for the RSPI transmission/reception.

Initializes the RSPI channel 4 to set the bit rate at 2.78Mbps when P1_f is 66.67 MHz.

Sets the "Block Protect Bits" in the EEPROM status register to B'00 to release the protection for the entire EEPROM area.

Starting from EEPROM memory address H'0000, transmits 16KB of data from the transmit buffer trans_data[].

Starting from EEPROM memory address H'0000, receives 16KB of data and stores it in the receive buffer receive_data[].

Sets the "Block Protect Bits" in the EEPROM status register to B'11 to set protection to the entire EEPROM area.
5.9.5 PORT Initial Setting Function

Figure 5.11 shows the flowchart of PORT Initial Setting Function.

In the sample code, P4_0 pin, P4_1 pin, P4_2 pin, and P4_3 pin of the PORT is set to perform the functions as RSPCK4, SSL40, MOSI4, and MISO4 respectively.

![Flowchart of PORT Initial Setting Function](image)

To enable transfers on RSPI channel 4, multiplexed pin functions are set to use P4_0 pin, P4_1 pin, P4_2 pin, and P4_3 pin as RSPCK4, SSL40, MOSI4, and MISO4 respectively.
5.9.6 Sample Processing to Set or Release Protection for EEPROM

Figure 5.12 shows the flowchart of Sample Processing to Set or Release Protection for EEPROM.

When the argument status indicates write protection release, writes "H'00" to the EEPROM status register to release write protection for the entire EEPROM area (bits BP1 and BP0 both cleared to "0").

When the argument status indicates write protection, writes "H'0C" to the EEPROM status register to set write protection for the entire EEPROM area (bits BP1 and BP0 both set to "1").
5.9.7 Sample Processing for Writing to EEPROM

Figure 5.13 shows the flowchart of Sample Processing for Writing to EEPROM.

```plaintext
Sample_RSPI_EepWrite_8bit
  Set EEPROM WRITE command and memory address
    cmd[0] ← H'02,
    cmd[1] ← (uint8_t)(addr >> 8), cmd[2] ← (uint8_t)(addr & 0xFF)
    Stores the write command and the memory address generated from argument addr
    in the transmit buffer.
    Transmits instruction code "WREN (H'06)".
    Processing for data transmission to EEPROM
    RSPI_EepWriteEnable_8bit()
  EEPROM write-enable processing
    RSPI_EepWriteEnable_8bit()
    Transmits the instruction code "WRITE (H'02)" specified by the argument cmd and the
    memory address, then transmits the byte count specified by the argument byte stored
    in the area specified by the argument buffer starting with the said memory address.
    In the sample code, the 16KB data is written using the 128-byte page write.
  Waiting for completion of EEPROM write processing
    RSPI_EepBusyWait_8bit()
    Transmits the instruction code "RDSR (H'05)" and waits until the data has been written
    into the EEPROM.
  return
```

Figure 5.13  Sample Processing for Writing to EEPROM

5.9.8 Sample Processing for Reading from EEPROM

Figure 5.14 shows the flowchart of Sample Processing for Reading from EEPROM.

```plaintext
Sample_RSPI_EepRead_8bit
  Set EEPROM READ command and memory address
    cmd[0] ← H'03,
    cmd[1] ← (uint8_t)(addr >> 8), cmd[2] ← (uint8_t)(addr & 0xFF)
    Stores the read command and the memory address generated from argument addr
    in the transmit buffer.
    Transmits the instruction code "READ (H'03)" specified by the argument cmd and the
    memory address, then receives the data for the byte count specified by the argument
    byte starting with the said memory address. The data is stored in the area specified by
    the argument buffer.
  Processing for data reception from EEPROM
    RSPI_EepRead()
  return
```

Figure 5.14  Sample Processing for Reading from EEPROM
5.9.9 RSPI Channel 4 Transmit Buffer Empty Interrupt Processing

Figure 5.15 shows the flowchart of RSPI Channel 4 Transmit Buffer Empty Interrupt Processing.

In the sample code, this function is registered as the INTC handler by using the user-defined function Userdef_RSPI4_Init.

![Diagram of RSPI Channel 4 Transmit Buffer Empty Interrupt Processing]

5.9.10 RSPI Channel 4 Receive Buffer Full Interrupt Processing

Figure 5.16 shows the flowchart of RSPI Channel 4 Receive Buffer Full Interrupt Processing.

In the sample code, this function is registered as the INTC handler by using the user-defined function Userdef_RSPI4_Init.

![Diagram of RSPI Channel 4 Receive Buffer Full Interrupt Processing]
5.9.11  EEPROM Status Write Processing

Figure 5.17 shows the flowchart of EEPROM Status Write Processing.

```plaintext
RSPI_EepWriteStatus_8bit

Set EEPROM WRSR command

cmd[0] ← H'01,
cmd[1] ← status
Stores the WRSR command and argument status in the transmit buffer.

EEPROM write-enable processing
RSPI_EepWriteEnable_8bit()

Transmits instruction code "WREN (H'06)".

Processing for data transmission to EEPROM
RSPI_EepWrite()

Transmits the instruction code "WRSR (H'01)" specified by the argument cmd and the value to be set in the status register.

Wait processing for completion of EEPROM write operation
RSPI_EepBusyWait_8bit()

Transmits the instruction code "RDSR (H'05)" and waits until the data has been written into the EEPROM.

return
```

Figure 5.17  EEPROM Status Write Processing

5.9.12  EEPROM Status Read Processing

Figure 5.18 shows the flowchart of EEPROM Status Read Processing.

```plaintext
RSPI_EepReadStatus_8bit

Set EEPROM RDSR command

cmd[0] ← H'05
Stores the RDSR command in the transmit data buffer.

Processing for data reception from EEPROM
RSPI_EepRead()

buffer ← Value read from status register
Transmits the instruction code "RDSR (H'05)" specified by the argument cmd.
Receives the status register value and returns it as a return value.

return (buffer)
```

Figure 5.18  EEPROM Status Read Processing
5.9.13  EEPROM Write-Enable Processing

Figure 5.19 shows the flowchart of EEPROM Write-Enable Processing.

![Diagram of EEPROM Write-Enable Processing]

- **RSPI_EepWriteEnable_8bit**
- **Set EEPROM WREN command**
  - cmd[0] ← H'06
  - Stores the WREN command in the transmit data buffer.
- **Processing for data transmission to EEPROM RSPI_EepWrite()**
  - Transmits the instruction code "WREN (H'06)" specified by the argument cmd.
- **return**

Figure 5.19  EEPROM Write-Enable Processing

5.9.14  Wait Processing for Completion of EEPROM Write Operation

Figure 5.20 shows the flowchart of Wait Processing for Completion of EEPROM Write Operation.

![Diagram of Wait Processing for Completion of EEPROM Write Operation]

- **RSPI_EepBusyWait_8bit**
- **EEPROM status read processing RSPI_EepReadStatus_8bit()**
  - Reads the EEPROM status register.
  - The EEPROM write operation is in progress if "1" is specified for the value of the WIP bit in the read status register, so the function waits until its value turns to "0".
- **return**

Figure 5.20  Wait Processing for Completion of EEPROM Write Operation
5.9.15 Processing for Data Transmission to EEPROM

Figure 5.21 shows the flowchart of Processing for Data Transmission to EEPROM.

```
RSPI_EepWrite
    Reset RSPI buffer
    R_RSPI_ResetBuffer()

RSPI function enable settings
    R_RSPI_Open()

Instruction code transmission?
    Argument ope_count = 0
    Argument ope_count = 1 or more

Transmit instruction code
    R_RSPI_Write()

Data transmission?
    Argument w_data_count = 0
    Argument w_data_count

Transmit write data
    R_RSPI_Write()

Wait for RSPI end of transmission
    R_RSPI_WaitTransmitEnd()

RSPI function disable settings
    R_RSPI_Close()

return
```

Transmits the instruction code specified by the argument ope_data and the memory address for the counts specified by the argument ope_count using the RSPI channel 4. Changes the transmit buffer data triggering number to 7 bytes, and sets the conditions for the transmit buffer empty occurrence notification to be satisfied when empty space in the transmit buffer is 1 byte or more.

Transmits the transmit data specified by the argument w_data for the counts specified by the argument w_data_count using the RSPI channel 4. The transmit buffer data triggering number operates according to the value of the TXTRG bits in the initial settings.

Waits for the RSPI channel 4 end of transmission.

Disables the RSPI channel 4 function and negates the SSL signal.

Figure 5.21  Processing for Data Transmission to EEPROM
5.9.16 Processing for Data Reception from EEPROM

Figure 5.22 shows the flowchart of Processing for Data Reception from EEPROM.

```
RSPI_EepRead

Reset RSPI buffer
R_RSPI_ResetBuffer()

RSPI function enable settings
R_RSPI_Open()

Instruction code transmission?

Argument ope_count = 0

Transmit instruction code
R_RSPI_Write()

Wait for RSPI end of transmission
R_RSPI_WaitTransmitEnd()

Argument ope_count = 1 or more

Receive data ≥ data count specified by RXTRG bits?

Equal to or larger than data count specified by RXTRG bits

Set information indicating no change in RXTRG bits to variable r_byte_flg

Receive read data
R_RSPI_Read()

Wait for end of dummy data transmission
R_RSPI_WaitTransmitEnd()

RSPI function disable settings
R_RSPI_Close()

Argument ope_count = 1 or more

Receive data < data count specified by RXTRG bits

Less than data count specified by RXTRG bits

Set information indicating change in RXTRG bits to variable r_byte_flg

return
```

Figure 5.22 Processing for Data Reception from EEPROM

- Resets the RSPI channel 4 transmit buffer and receive buffer.
- Enables RSPI channel 4 function.
- Transmits the instruction code specified by the argument ope_data and the memory address for the counts specified by the argument ope_count using the RSPI channel 4.
- Changes the transmit buffer data triggering number to 7 bytes and sets the conditions for the transmit buffer empty occurrence notification to be satisfied immediately when empty space in the transmit buffer is 1 byte or larger.
- Waits for satisfaction of the conditions for RSPI channel 4 end of transmission.
- Resets the transmit and receive buffers of RSPI channel 4.
- Receives the data for the counts specified by the argument r_data_count and stores it in the area specified by the argument r_data using the RSPI channel 4.
- Changes the setting of the RXTRG bits according to the contents of the argument r_byte_flg. If the received data count is less than data count specified by the RXTRG bits, these bits are set to "B'000" so that a receive buffer-full interrupt is generated each time one byte of data is stored in the buffer. If the received data count is equal to or larger than the data count specified by the RXTRG bits, the setting of the RXTRG bits is not changed.
- Waits until the conditions are satisfied, which set out for end of transmission of the dummy data transmitted by the RSPI channel 4 in order to generate a clock when the data is received.
- Disables the RSPI channel 4 function and negates the SSL signal.
5.9.17  RSPI Initial Settings

Figure 5.23 shows the flowchart of RSPI Initial Settings.

Figure 5.23   RSPI Initial Settings
5.9.18 RSPI Function Enable Settings

Figure 5.24 shows the flowchart of RSPI Function Enable Settings.

```
R_RSPI_Open

Argument error in function?

Error in specification for channel

return (DEVDRV_ERROR)

Channel?

channel = 0

Enable functionality of RSPI channel 0

SPCR_0 register
SPE bit ← 1 : Enables RSPI functionality.

channel = 1

Enable functionality of RSPI channel 1

SPCR_1 register
SPE bit ← 1 : Enables RSPI functionality.

channel = 2

Enable functionality of RSPI channel 2

SPCR_2 register
SPE bit ← 1 : Enables RSPI functionality.

channel = 3

Enable functionality of RSPI channel 3

SPCR_3 register
SPE bit ← 1 : Enables RSPI functionality.

channel = 4

Enable functionality of RSPI channel 4

SPCR_4 register
SPE bit ← 1 : Enables RSPI functionality.

default

return (DEVDRV_SUCCESS)
```

Figure 5.24 RSPI Function Enable Settings
5.9.19 RSPI Function Disable Settings

Figure 5.25 shows the flowchart of RSPI Function Disable Settings.

Figure 5.25 RSPI Function Disable Settings
5.9.20 RSPI Transmit Processing

Figure 5.26 to Figure 5.29 show the flowcharts of RSPI Transmit Processing.

- **R_RSPI_Write**
  - Error in specification for channel and txtrg
    - return (DEVDRV_ERROR)
  - Argument error in function?
    - return (DEVDRV_ERROR)
  - Specified transmission count 1 or more?
    - return (DEVDRV_SUCCESS)
    - Argument count = 0
    - Argument count = 1 or more
    - Change in TXTRG bits?
      - No change in TXTRG bits
      - Change in TXTRG bits
  - Read setting value of transmit buffer data triggering number
    - SPBFCR register
      - txtrg_backup ← TXTRG bit
        : Reads and saves the value of TXTRG bits.
  - Write value specified by argument to transmit buffer data triggering number
    - SPBFCR register
      - TXTRG ← argument txtrg
        : Changes the value of the TXTRG bits to the specified triggering number.
      - In the sample code, the TXTRG bits are set to 1 byte, and a transmit buffer empty interrupt is generated every 1-byte transmission when the transmit data count such as transmission of instruction code is small.
  - Obtain access width setting information
    - Reads SPDCR register.
      - SPLW bit:
        1: SPDR register access width is 1 byte.
        2: SPDR register access width is 2 bytes.
        3: SPDR register access width is 4 bytes.
  - Obtain transmit buffer data triggering number information
    - Reads SPBFCR register
      - TXTRG bit:
        0: Empty when the empty byte count in transmit buffer is 1
        1: Empty when the empty byte count in transmit buffer is 2
        2: Empty when the empty byte count in transmit buffer is 4
        3: Empty when the empty byte count in transmit buffer is 8
      - Specified transmit byte count > byte count specified by SPLW bits?
        - Byte count specified by TXTRG bits < setting of SPLW bits
          - return (DEVDRV_ERROR)
        - Specified transmit byte count is integer multiple of byte count specified by SPLW bits?
          - (Byte count specified by TXTRG bits % setting of SPLW bits) \(!= 0\)
            - return (DEVDRV_ERROR)

---

Figure 5.26 RSPI Transmit Processing (1/4)
Transmission count is 1 or larger?

Set transmit buffer data triggering number as transmit data count

channel = 0

Wait for satisfaction of the conditions for RSPI channel 0 transmit buffer empty notification information
Userdef_RSPI0_WaitTransmitEmpty()

channel = 1

Wait for satisfaction of the conditions for RSPI channel 1 transmit buffer empty notification information
Userdef_RSPI1_WaitTransmitEmpty()

channel = 2

Wait for satisfaction of the conditions for RSPI channel 2 transmit buffer empty notification information
Userdef_RSPI2_WaitTransmitEmpty()

channel = 3

Wait for satisfaction of the conditions for RSPI channel 3 transmit buffer empty notification information
Userdef_RSPI3_WaitTransmitEmpty()

channel = 4

Wait for satisfaction of the conditions for RSPI channel 4 transmit buffer empty notification information
Userdef_RSPI4_WaitTransmitEmpty()

default

Byte count specified by argument count has been transmitted

Byte count specified by argument count should be transmitted.

Waits until the conditions for the RSPI transmit buffer empty notification information to be satisfied.

In the sample code, this function enables the RSPI channel 4 transmit interrupt (SPTI4) by using the user-defined function Userdef_RSPI4_WaitTransmitEmpty and waits until the transmit buffer empty notification flag is set.

A user-defined function has been prepared for each of the RSPI channels, but the following functions are not used in the sample code.
Userdef_RSPI0_WaitTransmitEmpty
Userdef_RSPI1_WaitTransmitEmpty
Userdef_RSPI2_WaitTransmitEmpty
Userdef_RSPI3_WaitTransmitEmpty

Figure 5.27 RSPI Transmit Processing (2/4)
Transmission count and transmit data count are both 1 or larger?

Data has been transmitted

Data should be transmitted

Write data to transmit buffer

SPDR register ← buffer
Writes the data in the area specified by the argument buffer to the transmit buffer.

Receive buffer full?

Receive buffer full

Space in receive buffer

Perform dummy read of 1 data byte to prevent receive data overflow

Performs dummy read of SPDR register.

Decrement transmission count and transmit data count

Decrement the transmission count specified by the argument and decrements the transmit data count by the access width byte count.

Transmits the data for the transmit buffer data triggering number.
The data is written to the transmit buffer using the access width acquired from the SPLW bits.

Performs a dummy read of receive data to suspend the transfer when empty space for the receive data length is not provided in the receive buffer.

Figure 5.28  RSPI Transmit Processing (3/4)
channel = 0
Initialize RSPI channel 0 transmit buffer empty notification information
Userdef_RSPI0_InitTransmitEmpty()

channel = 1
Initialize RSPI channel 1 transmit buffer empty notification information
Userdef_RSPI1_InitTransmitEmpty()

channel = 2
Initialize RSPI channel 2 transmit buffer empty notification information
Userdef_RSPI2_InitTransmitEmpty()

channel = 3
Initialize RSPI channel 3 transmit buffer empty notification information
Userdef_RSPI3_InitTransmitEmpty()

channel = 4
Initialize RSPI channel 4 transmit buffer empty notification information
Userdef_RSPI4_InitTransmitEmpty()

default

Figure 5.29   RSPI Transmit Processing (4/4)
5.9.21 RSPI Receive Processing

Figure 5.30 to Figure 5.32 show the flowcharts of RSPI Receive Processing.

### RSPI Receive Processing

- **R_RSPI_Read**
- **Argument error on function?**
  - return (DEVDRV_ERROR)
- **Specified reception count is 1 or more?**
  - Argument count = 0
  - return (DEVDRV_SUCCESS)
- **Change in RXTRG bits?**
  - Change in RXTRG bits
  - Read setting value of receive buffer data triggering number
  - SPBFCR register
    - rxtrg_backup ← RXTRG bit
    - : Reads and saves the value of RXTRG bits
  - Write value specified by argument to receive buffer data triggering number
  - SPBFCR register
    - RXTRG ← Argument rxtrg
    - : Changes the value of the RXTRG bits to the specified triggering count.
    - In the sample code, the RXTRG bits are set to 1 byte, and a receive buffer full interrupt is generated every 1-byte reception when the receive data count such as reception of EEPROM status is small.
- **Obtain access width setting information**
  - Reads SPDCR register.
    - SPLW bit:
      1: SPDR register access width is 1 byte.
      2: SPDR register access width is 2 bytes.
      3: SPDR register access width is 4 bytes.
- **Obtain receive buffer data triggering number information**
  - Read SPBFCR register
    - RXTRG bit:
      0: Full when the byte count in receive buffer is 1
      1: Full when the byte count in receive buffer is 2
      2: Full when the byte count in receive buffer is 4
      3: Full when the byte count in receive buffer is 8
      4: Full when the byte count in receive buffer is 16
      5: Full when the byte count in receive buffer is 24
      6: Full when the byte count in receive buffer is 32
      7: Full when the byte count in receive buffer is 64
- **Specified receive byte count ≥ byte count specified by SPLW bits?**
  - Byte count specified by RXTRG bits < setting of SPLW bits
  - return (DEVDRV_ERROR)
- **Specified receive byte count is integer multiple of byte count specified by SPLW bits?**
  - (Byte count specified by RXTRG bits % setting of SPLW bits) ≠ 0
  - return (DEVDRV_ERROR)
Reception count is 1 or larger?

Reception necessary for byte count specified by argument count

Reception count < receive buffer data triggering number

Channel = 4

Default

Receive-end for byte count specified by argument count

Channel = 2

Channel = 3

Channel = 1

Channel = 0

Wait for satisfaction of the conditions for RSPI channel 0 notification information

Userdef_RSPI0_WaitReceiveFull()

Wait for satisfaction of the conditions for RSPI channel 1 notification information

Userdef_RSPI1_WaitReceiveFull()

Wait for satisfaction of the conditions for RSPI channel 2 notification information

Userdef_RSPI2_WaitReceiveFull()

Wait for satisfaction of the conditions for RSPI channel 3 notification information

Userdef_RSPI3_WaitReceiveFull()

Wait for satisfaction of the conditions for RSPI channel 4 notification information

Userdef_RSPI4_WaitReceiveFull()

Set reception count to receive data count

Set byte count of receive buffer data triggering number to receive data count

Waits for the conditions for the RSPI receive buffer full notification information to be satisfied.

In the sample code, this function enables the RSPI channel 4 receive interrupt (SPRI4) by using the user-defined function Userdef_RSPI4_WaitReceiveFull and waits until the receive buffer full notification flag is set.

A user-defined function has been prepared for each of the RSPI channels, but the following functions are not used in the sample code.

Userdef_RSPI0_WaitReceiveFull
Userdef_RSPI1_WaitReceiveFull
Userdef_RSPI2_WaitReceiveFull
Userdef_RSPI3_WaitReceiveFull

Reception count is less than receive buffer triggering number?

Argument count < receive buffer data triggering number

Argument count ≥ receive buffer data triggering number

Figure 5.31 RSPI Receive Processing (2/3)
Receive data count is 1 or larger?

Data should be received

*buffer ← SPDR register

Writes the data read from the receive buffer to the area specified by the argument buffer.

Decrement reception count and receive data count

Read data from receive buffer

Decrement the receive data count by the access width byte count.

channel = 0

Initialize RSPI channel 0 receive buffer full notification information
Userdef_RSPI0_InitReceiveFull()

channel = 1

Initialize RSPI channel 1 receive buffer full notification information
Userdef_RSPI1_InitReceiveFull()

channel = 2

Initialize RSPI channel 2 receive buffer full notification information
Userdef_RSPI2_InitReceiveFull()

channel = 3

Initialize RSPI channel 3 receive buffer full notification information
Userdef_RSPI3_InitReceiveFull()

channel = 4

Initialize RSPI channel 4 receive buffer full notification information
Userdef_RSPI4_InitReceiveFull()

default

Trigger not changed

RXTRG bits changed?

Trigger has been changed

Restore setting value of saved receive buffer data triggering number

SPBFCR register
RXTRG bit ← nxtrg_backup
Restores the RXTRG bits to the setting value before the change.

return (DEVDRV_SUCCESS)

Reads from the receive buffer using the access width acquired from the SPLW bits.

Initializes the RSPI receive buffer full notification information.

In the sample code, the RSPI channel 4 receive buffer full notification flag is cleared by using the user-defined function Userdef_RSPI4_InitReceiveFull.

A user-defined function has been prepared for each of the RSPI channels, but the following functions are not used in the sample code.
Userdef_RSPI0_InitReceiveFull
Userdef_RSPI1_InitReceiveFull
Userdef_RSPI2_InitReceiveFull
Userdef_RSPI3_InitReceiveFull

Figure 5.32 RSPI Receive Processing (3/3)
5.9.22 RSPI Buffer Reset Processing

Figure 5.33 and Figure 5.34 show the flowcharts of RSPI Buffer Reset Processing.

![Flowchart of RSPI Buffer Reset Processing](image)

In the sample code, the transmit buffer empty and the receive buffer full notification flags of the RSPI channel 4 are cleared by using the user-defined functions Userdef_RSPI4_InitTransmitEmpty and Userdef_RSPI4_InitReceiveFull.

![Flowchart of RSPI Buffer Reset Processing](image)

A user-defined function has been prepared for each of the RSPI channels, but the following functions are not used in the sample code.

Userdef_RSPI0_InitTransmitEmpty
Userdef_RSPI0_InitReceiveFull
Userdef_RSPI1_InitTransmitEmpty
Userdef_RSPI1_InitReceiveFull
Userdef_RSPI2_InitTransmitEmpty
Userdef_RSPI2_InitReceiveFull
Userdef_RSPI3_InitTransmitEmpty
Userdef_RSPI3_InitReceiveFull
Figure 5.34 RSPI Buffer Reset Processing (2/2)
5.9.23 Wait Processing for RSPI End of Transmission

Figure 5.35 shows the flowchart of Wait Processing for RSPI End of Transmission.

```
R_RSPI_WaitTransmitEnd

Argument error in function?

return (DEVDRV_ERROR)

Channel?

channel = 0

Wait for satisfaction of the conditions
for RSPI channel 0
end of transmission
Userdef_RSPI0_WaitTransmitEnd()

channel = 1

Wait for satisfaction of the conditions
for RSPI channel 1
end of transmission
Userdef_RSPI1_WaitTransmitEnd()

channel = 2

Wait for satisfaction of the conditions
for RSPI channel 2
end of transmission
Userdef_RSPI2_WaitTransmitEnd()

channel = 3

Wait for satisfaction of the conditions
for RSPI channel 3
end of transmission
Userdef_RSPI3_WaitTransmitEnd()

channel = 4

Wait for satisfaction of the conditions
for RSPI channel 4
end of transmission
Userdef_RSPI4_WaitTransmitEnd()

default

return (DEVDRV_SUCCESS)
```

Waits for RSPI end of transmission.

In the sample code, this function waits for the RSPI channel 4 end of transmission by polling the TEND bit in the SPSR_4 register using the user-defined function Userdef_RSPI4_WaitTransmitEnd.

A user-defined function has been prepared for each of the RSPI channels, but the following functions are not used in the sample code.

Userdef_RSPI0_WaitTransmitEnd
Userdef_RSPI1_WaitTransmitEnd
Userdef_RSPI2_WaitTransmitEnd
Userdef_RSPI3_WaitTransmitEnd

Figure 5.35 Wait Processing for RSPI End of Transmission
5.9.24 RSPI Transmit Buffer Empty Interrupt Notification Processing

Figure 5.36 shows the flowchart of RSPI Transmit Buffer Empty Interrupt Notification Processing.

```
R_RSPI_SptiInterrupt
Argument error in function?
return (DEVDRV_ERROR)
Channel?
channel = 0
Setting to satisfy the conditions for RSPI channel 0 transmit buffer empty notification information
Userdef_RSPI0_SetTransmitEmpty()
channel = 1
Setting to satisfy the conditions for RSPI channel 1 transmit buffer empty notification information
Userdef_RSPI1_SetTransmitEmpty()
channel = 2
Setting to satisfy the conditions for RSPI channel 2 transmit buffer empty notification information
Userdef_RSPI2_SetTransmitEmpty()
channel = 3
Setting to satisfy the conditions for RSPI channel 3 transmit buffer empty notification information
Userdef_RSPI3_SetTransmitEmpty()
channel = 4
Setting to satisfy the conditions for RSPI channel 4 transmit buffer empty notification information
Userdef_RSPI4_SetTransmitEmpty()
default
return (DEVDRV_SUCCESS)
```

Makes settings to satisfy the conditions for the RSPI transmit buffer empty notification information.

In the sample code, this function clears the SPTIE bit in the SPCR_4 register by using the user-defined function Userdef_RSPI4_SetTransmitEmpty and disables the RSPI channel 4 transmit interrupt (SPTI4). Then sets the flag which notifies that the transmit buffer empty interrupt has been generated.

A user-defined function has been prepared for each of the RSPI channels, but the following functions are not used in the sample code.
Userdef_RSPI0_SetTransmitEmpty
Userdef_RSPI1_SetTransmitEmpty
Userdef_RSPI2_SetTransmitEmpty
Userdef_RSPI3_SetTransmitEmpty

Figure 5.36 RSPI Transmit Buffer Empty Interrupt Notification Processing
5.9.25 RSPI Receive Buffer Full Interrupt Notification Processing

Figure 5.37 shows the flowchart of RSPI Receive Buffer Full Interrupt Notification Processing.

![Flowchart of RSPI Receive Buffer Full Interrupt Notification Processing]

- **R_RSPI_SpriInterrupt**
  - **Argument error in function?**
    - **return (DEVDRV_ERROR)**
  - **Channel?**
    - channel = 0
      - Setting to satisfy the conditions for RSPI channel 0 receive buffer full notification information
        - Userdef_RSPI0_SetReceiveFull()
    - channel = 1
      - Setting to satisfy the conditions for RSPI channel 1 receive buffer full notification information
        - Userdef_RSPI1_SetReceiveFull()
    - channel = 2
      - Setting to satisfy the conditions for RSPI channel 2 receive buffer full notification information
        - Userdef_RSPI2_SetReceiveFull()
    - channel = 3
      - Setting to satisfy the conditions for RSPI channel 3 receive buffer full notification information
        - Userdef_RSPI3_SetReceiveFull()
    - channel = 4
      - Setting to satisfy the conditions for RSPI channel 4 receive buffer full notification information
        - Userdef_RSPI4_SetReceiveFull()
    - default
      - **return (DEVDRV_SUCCESS)**

Initializes the RSPI receive buffer full notification information.

In the sample code, this function clears the SPRIE bit in the SPCR_4 register and disables the RSPI channel 4 receive interrupt (SPRI4). Then sets the flag which notifies that the receive buffer full interrupt has been generated.

A user-defined function has been prepared for each of the RSPI channels, but the following functions are not used in the sample code.

- Userdef_RSPI0_SetReceiveFull
- Userdef_RSPI1_SetReceiveFull
- Userdef_RSPI2_SetReceiveFull
- Userdef_RSPI3_SetReceiveFull
- Userdef_RSPI4_SetReceiveFull

Figure 5.37 RSPI Receive Buffer Full Interrupt Notification Processing
5.9.26 RSPI Channel 4 Initial Setting Function

Figure 5.38 and Figure 5.39 show the flowcharts of RSPI Channel 4 Initial Setting Function. The API functions are used to register the interrupt function of the RSPI channel 4 as the INTC handler, to set the interrupt priority level, and to enable the interrupts. Refer to the RZ/A1H group application note for Example of Initialization Application for more details about the API functions for the INTC interrupt; R_INTC_RegistIntFunc, R_INTC_SetPriority, and R_INTC_Enable.

User-defined functions for other channels of the RSPI are prepared, but these are not use in the sample code. When other channel except channel 4 is applied, the function of the channel applied should be executed to make initial settings.

![Flowchart of RSPI Channel 4 Initial Setting Function](image-url)

**Figure 5.38 RSPI Channel 4 Initial Setting Function (1/2)**
### RSPI Channel 4 Initial Setting Function (2/2)

**Reset transmit and receive buffers and set buffer data triggering number**

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPBFCR_4</td>
<td>H'F5</td>
<td>Reset transmit and receive buffers, set buffer data triggering number.</td>
</tr>
</tbody>
</table>

- TXRST bit = 1 : Enables transmit buffer reset operation.
- RXRST bit = 1 : Enables receive buffer reset operation.
- TXTRG bit = 3 : Sets the transmit buffer to become empty when data stored in transmit buffer reaches 0 bytes or less. (8 bytes for empty space in buffer)
- RXTRG bit = 5 : Sets the receive buffer to become full when data stored in receive buffer reaches to 24 bytes or more.

**Set SSL polarity**

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLP_4</td>
<td>H'00</td>
<td>Set SSL signal active polarity to &quot;0&quot;.</td>
</tr>
</tbody>
</table>

**Set master mode and disable interrupt requests**

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPCR_4</td>
<td>H'08</td>
<td>Set master mode, disable interrupt requests.</td>
</tr>
</tbody>
</table>

- SPRIE bit = 0 : Disables receive interrupt request.
- SPE bit = 0 : Disables RSPI functionality.
- SPTIE bit = 0 : Disables transmit interrupt request.
- SPEIE bit = 0 : Disables error interrupt request.
- MSTR bit = 1 : Specifies master mode.
- MODFEN bit = 0 : Disables mode fault error detection.

**Reset release for transmit and receive buffers**

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPBFCR_4</td>
<td>H'3F</td>
<td>Reset release for transmit and receive buffers.</td>
</tr>
</tbody>
</table>

- TXRST bit = 0 : Disables transmit buffer reset operation.
- RXRST bit = 0 : Disables receive buffer reset operation.

**Register SPTI4 interrupt handler**

R_INTC_RegistIntFunc()

**Set SPTI4 interrupt priority level**

R_INTC_SetPriority()

**Enable SPTI4 interrupt**

R_INTC_Enable()

**Register SPR4 interrupt handler**

R_INTC_RegistIntFunc()

**Set SPR4 interrupt priority level**

R_INTC_SetPriority()

**Enable SPR4 interrupt**

R_INTC_Enable()

---

**Figure 5.39 RSPI Channel 4 Initial Setting Function (2/2)**
5.9.27 Initialization Function for RSPI Channel 4 Transmit Buffer Empty Notification Information

Figure 5.40 shows the flowchart of Initialization Function for RSPI Channel 4 Transmit Buffer Empty Notification Information.

User-defined functions for other channels of the RSPI are prepared, but these are not use in the sample code. When other channel except channel 4 is applied, the function of the channel applied should be executed to initialize the transmit buffer empty notification information.

Userdef_RSPI4_InitTransmitEmpty

Clear RSPI channel 4 transmit buffer empty notification flag

Clears the flag used to notify that the RSPI channel 4 transmit buffer empty interrupt has been generated.

return

Figure 5.40 Initialization Function for RSPI Channel 4 Transmit Buffer Empty Notification Information

5.9.28 Initialization Function for RSPI Channel 4 Receive Buffer Full Notification Information

Figure 5.41 shows the flowchart of Initialization Function for RSPI Channel 4 Receive Buffer Full Notification Information.

User-defined functions for other channels of the RSPI are prepared, but these are not use in the sample code. When other channel except channel 4 is applied, the function of the channel applied should be executed to initialize the receive buffer full notification information.

Userdef_RSPI4_InitReceiveFull

Clear RSPI channel 4 receive buffer full notification flag

Clears the flag used to notify that the RSPI channel 4 receive buffer full interrupt has been generated.

return

Figure 5.41 Initialization Function for RSPI Channel 4 Receive Buffer Full Notification Information
5.9.29 Setting Function to Satisfy the Conditions for RSPI Channel 4 Transmit Buffer Empty Notification Information

Figure 5.42 shows the flowchart of Setting Function to Satisfy the Conditions for RSPI Channel 4 Transmit Buffer Empty Notification Information.

User-defined functions for other channels of the RSPI are prepared, but these are not use in the sample code. When other channel except channel 4 is applied, the function of the channel applied should be executed to satisfy conditions for the transmit buffer empty notification information.

```
Userdef_RSPI4_SetTransmitEmpty
```

```
Disable RSPI channel 4 transmit interrupt requests
```

```
Set the RSPI channel 4 transmit buffer empty notification flag
```

```
SPCR_4 register
SPTIE bit ← 0 : Enables the RSPI channel 4 transmit interrupt requests.
```

Sets the flag used to notify that the RSPI channel 4 transmit buffer empty interrupt has been generated.

Figure 5.42 Setting Function to Satisfy the Conditions for RSPI Channel 4 Transmit Buffer Empty Notification Information

5.9.30 Setting Function to Satisfy the Conditions for RSPI Channel 4 Receive Buffer Full Notification Information

Figure 5.43 shows the flowchart of Setting Function to Satisfy the Conditions for RSPI Channel 4 Receive Buffer Full Notification Information.

User-defined functions for other channels of the RSPI are prepared, but these are not use in the sample code. When other channel except channel 4 is applied, the function of the channel applied should be executed to satisfy conditions for the receive buffer full notification information.

```
Userdef_RSPI4_SetReceiveFull
```

```
Disable RSPI channel 4 receive interrupt requests
```

```
Set the RSPI channel 4 receive buffer full notification flag
```

```
SPCR_4 register
SPRIE bit ← 0 : Disables the RSPI channel 4 receive interrupt requests.
```

Sets the flag used to notify that the RSPI channel 4 receive buffer full interrupt has been generated.

Figure 5.43 Setting Function to Satisfy the Conditions for RSPI Channel 4 Receive Buffer Full Notification Information
5.9.31 Wait Function for Satisfaction of the Conditions for RSPI Channel 4 Transmit Buffer Empty Notification Information

Figure 5.44 shows the flowchart of Wait Function for Satisfaction of the Conditions for RSPI Channel 4 Transmit Buffer Empty Notification Information.

User-defined functions for other channels of the RSPI are prepared, but these are not used in the sample code. When other channel except channel 4 is applied, the function of the channel applied should be executed to wait until the conditions for the transmit buffer empty notification information to be satisfied.

```
Userdef_RSPI4_WaitTransmitEmpty

Enable RSPI channel 4 transmit interrupt requests

Wait until RSPI channel 4 transmit buffer empty notification flag to be set

SPCR_4 register
SPTIE bit ← 1 : Enables the RSPI channel 4 transmit interrupt requests.

Wait until the flag is set to notify that the RSPI channel 4 transmit buffer empty interrupt has been generated.

return
```

Figure 5.44 Wait Function for Satisfaction of the Conditions for RSPI Channel 4 Transmit Buffer Empty Notification Information

5.9.32 Wait Function for Satisfaction of the Conditions for RSPI Channel 4 End of Transmission

Figure 5.45 shows the flowchart of Wait Function for Satisfaction of the Conditions for RSPI Channel 4 End of Transmission.

User-defined functions for other channels of the RSPI are prepared, but these are not used in the sample code. When other channel except channel 4 is applied, the function of the channel applied should be executed to wait until the conditions for the end of transmission to be satisfied.

```
Userdef_RSPI4_WaitTransmitEnd

Wait for RSPI channel 4 end of transmission

Reads the SPSR_4 register.
Waits until the TEND bit is set to "1" : Waits for end of transmission.

return
```

Figure 5.45 Wait Function for Satisfaction of the Conditions for RSPI Channel 4 End of Transmission
5.9.33 Wait Function for Satisfaction of the Conditions for RSPI Channel 4 Receive Buffer Full Notification

Figure 5.46 and Figure 5.47 show the flowcharts of Wait Function for Satisfaction of the Conditions for RSPI Channel 4 Receive Buffer Full Notification.

User-defined functions for other channels of the RSPI are prepared, but these are not use in the sample code. When other channel except channel 4 is applied, the function of the channel applied should be executed to wait until the conditions for the receive buffer full notification information to be satisfied.

![Flowchart of Wait Function for Satisfaction of the Conditions for RSPI Channel 4 Receive Buffer Full Notification](image)

- **Userdef_RSPI4_WaitReceiveFull**
- **SPBFCR register**
  - btrg_backup ← TXTRG bit : Reads and saves value of TXTRG bits.
- **SPCR_4 register**
  - SPRIE bit ← 1 : Enables the RSPI channel 4 receive interrupt requests.
- **Obtain setting information of access width**
  - Reads SPDCR register
    - SPLW bit : 1: SPDR register access width is 1 byte.
      - 2: SPDR register access width is 2 bytes.
      - 3: SPDR register access width is 4 bytes.
- **Set value to TXTRG bits corresponding to access width**
  - Sets SPBFCR register
    - TXTRG bit ← 2 : Set when the access width is 4 bytes
    - TXTRG bit ← 1 : Set when the access width is 2 bytes
    - TXTRG bit ← 0 : Set when the access width is 1 byte
- **SPCR_4 register**
  - SPRIE bit ← 1 : Enables the RSPI channel 4 receive interrupt requests.
- **Enable RSPI channel 4 receive interrupt requests**

- **Receive data count is 1 byte or more?**
  - Byte count specified by the argument byte has been received
    - **Wait for satisfaction of the conditions for RSPI channel 4 transmit buffer empty notification information**
      - Userdef_RSPI4_WaitTransmitEmpty()
    - Byte count specified by the argument byte should be received
      - Enables the RSPI channel 4 transmit interrupt (SPTI4) requests and waits until the transmit buffer empty notification flag to be set.
- **Initialize RSPI channel 4 transmit buffer empty notification information**
  - Userdef_RSPI4_InitTransmitEmpty()
- **Write dummy data to transmit buffer**
  - Writes to the SPDR register.
    - When the access width is 4 bytes: Writes "H'FFFF FFFF".
    - When the access width is 2 bytes: Writes "H'FFFF".
    - When the access width is 1 byte: Writes "H'FF".
    - Write to the SPDR register to output the RSPCK clock for data reception.
- **Decrement receive data count**
  - Decrements the reception count specified by the argument for the byte count specified by the access width.

Figure 5.46 Wait Function for Satisfaction of the Conditions for RSPI Channel 4 Receive Buffer Full Notification (1/2)
Setting to satisfy the conditions for RSPI channel 4 receive buffer full notification information

Userdef_RSPI4_SetReceiveFull()

Argument less_rxtrg = "0"

Receive data count is less than receive buffer triggering number?

Argument less_rxtrg = "1"

(Receive data count is less than the count set by the RXTRG bits.)

Obtain byte count of received data

Obtain byte count of received data ≥ specified receive data count

Byte count of R[5:0] bits ≥ argument byte

Set the RSPI channel 4 receive buffer full notification flag.

Set the RSPI channel 4 receive buffer full notification flag.

Wait until the RSPI channel 4 receive buffer full notification flag to be set.

Wait for satisfaction of the conditions for RSPI channel 4 receive buffer full notification information

Restore setting value of saved transmit buffer data triggering number

SPBFCR register

TXTRG bit ← txtrg_backup

Restores the TXTRG bits to the setting value before the change.

Reads the SPBFDR register.

Obtains the byte count of received data with reference to the R[5:0] bits.

Receive data count is less than receive buffer triggering number?

Argument less_rxtrg = "0"

Argument less_rxtrg = "1"

Figure 5.47 Wait Function for Satisfaction of the Conditions for RSPI Channel 4 Receive Buffer Full Notification (2/2)
5.9.34 Period Securing Function to Start Next Access RSPI Channel 4

Figure 5.48 shows the flowchart of Period Securing Function to Start Next Access on RSPI Channel 4.

User-defined functions for other channels of the RSPI are prepared, but these are not use in the sample code. When other channel except channel 4 is applied, the function of the channel applied should be executed to secure the SSL signal negation period required to start the next access in accordance with the device connected.

```
Userdef_RSPI4_DelayNextAccess

Secure time period before start of next access on RSPI channel 4

return

Secures the SSL signal negation period required to start the next access in the communication with the RSPI channel 4.
In the sample code, the software wait processing is used to secure the SSL signal negation period (90ns or more) required for the EEPROM.
```

Figure 5.48 Period Securing Function to Start Next Access on RSPI Channel 4
5.10 Running the Sample Code

The sample code is operated by entering commands from the terminal software running on the host PC connected to the GENMAI board via the serial interface.

After supplying power to the GENMAI board, the message (1) shown in Figure 5.49 is output. To run the RSPI sample code, input "RSPI" + "Enter" key subsequent to the "SAMPLE>" prompt. When the message (2) shown in Figure 5.49 is output, input "1" + "Enter" key subsequent to the "RSPI SAMPLE>" prompt.

By inputting "HELP" + "Enter" key, the sample code information (3) shown in Figure 5.49 is displayed. "EXIT" + "Enter" key terminates the RSPI sample code operation.

Ver.X.XX and Ver.Y.YY shown in Figure 5.49 indicate the main processing version of sample code and the RSPI sample code version respectively.

<table>
<thead>
<tr>
<th>Display Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>RZ/A1H CPU Board Sample Program. Ver.X.XX</td>
</tr>
<tr>
<td>Copyright (C) 2015 Renesas Electronics Corporation. All rights reserved.</td>
</tr>
<tr>
<td>select sample program.</td>
</tr>
<tr>
<td>SAMPLE&gt;</td>
</tr>
</tbody>
</table>

| RZ/A1H Renesas Serial Peripheral Interface(RSPI) Sample Program. |
| Ver.Y.YY |
| Copyright (C) 2015 Renesas Electronics Corporation. All rights reserved. |
| select sample program. |
| RSPI SAMPLE> |

<table>
<thead>
<tr>
<th>RSPI SAMPLE&gt; help</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 : EEPROM access</td>
</tr>
<tr>
<td>- Bit rate : 2.78Mbps</td>
</tr>
<tr>
<td>EXIT : Exit RSPI sample</td>
</tr>
<tr>
<td>RSPI SAMPLE&gt;</td>
</tr>
</tbody>
</table>

Figure 5.49 Terminal Display at RSPI Sample Code Startup
6. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

7. Reference Documents

User's Manual: Hardware
   RZ/A1H Group User's Manual: Hardware
   The latest version can be downloaded from the Renesas Electronics website.

   The latest version can be downloaded from the Renesas Electronics website.

   The latest version can be downloaded from the Renesas Electronics website.

   ARM Generic Interrupt Controller Architecture Specification Architecture version 1.0
   The latest version can be downloaded from the Renesas Electronics website.

   ARM Cortex™-A9 (Revision: r3p0) Technical Reference Manual
   The latest version can be downloaded from the ARM website.

   ARM CoreLink™ Level 2 Cache Controller L2C-310 (Revision: r3p2) Technical Reference Manual
   The latest version can be downloaded from the ARM website.

   Technical Update/Technical News
   The latest information can be downloaded from the Renesas Electronics website.

   ARM software development tool (ARM Compiler toolchain, ARM DS-5, etc.) can be downloaded from the ARM website.
   The latest version can be downloaded from the ARM website.
Website and Support

Renesas Electronics website
http://www.renesas.com

Inquiries
http://www.renesas.com/contact/

All trademarks and registered trademarks are the property of their respective owners.
### Revision History

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rev.1.00</td>
<td>Jul. 11, 2014</td>
<td>-</td>
<td>First edition issued</td>
</tr>
<tr>
<td>Rev.1.01</td>
<td>Oct. 16, 2015</td>
<td></td>
<td>Reflected the update contents of &quot;RZ/A1H Group Example of Initialization Rev.1.01&quot; application note.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>P16</td>
<td>• Table 5.5 Sections Used (1/2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Changed the execution area for the CODE_IO_REGRW section in the table from &quot;FLASH&quot; to &quot;LRAM&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P17</td>
<td>• Table 5.6 Sections Used (2/2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added the CODE_CACHE_OPERATION section to the table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added precautions about the CODE_CACHE_OPERATION section to Note 3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P18</td>
<td>• Figure 5.5 Section Assignment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Provided information due to the CODE_IO_REGRW section arrangement being changed and the CODE_CACHE_OPERATION section being added.</td>
</tr>
</tbody>
</table>


General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins
   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.
   - The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   - The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable.
   When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   - When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal.
   Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.
   - The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.