RL78/G1E Group

Sample Code for Performing SPI Communication with Analog Block

R01AN1130EJ0120
Rev.1.20
Sep 30, 2013

Introduction

This application note describes the sample code used to access the SPI control registers by using the 3-wire serial I/O (CSI21) of channel 1 of serial array unit 1 incorporated in the RL78/G1E (R5F10FMx).

Target Devices

RL78/G1E (R5F10FMx (x = C, D, or E))

When the sample code shown in this application note is applied to other microcontrollers, make the necessary changes according to the specifications of the microcontroller and verify them thoroughly.

Contents

1. Specifications

1.1 Overview

This application note describes the sample code used to perform SPI communication with the analog block by using the 3-wire serial I/O of channel 1 of serial array unit 1 incorporated in the RL78/G1E (R5F10FMx).

The sample code used in this application note uses some of the serial interface functions generated by a code generator (CubeSuite+). Be sure to call the serial interface initialization function (auto code generation function) before using the sample code.

The serial interface functions (auto code generation functions) required to execute the sample code are shown below.

- `R_SAU1_Create`: Be sure to call this function before executing the sample code.
- `R_CSI21_Create`: This function is called from the `R_SAU1_Create` function.
- `R_CSI21_Start`: This function is called from the sample code.
- `R_CSI21_Stop`: This function is called from the sample code.
- `R_CSI21_SendReceive`: This function is called from the sample code.
- `r_csi21_interrupt`: This is the CSI21 interrupt handler.
- `r_csi21_callback_receiveend`: This function is called from the `r_csi21_interrupt` function.
- `r_csi21_callback_error`: This function is called from the `r_csi21_interrupt` function.

1.2 Procedure for using the sample code

How to use the sample code described in this application note is shown below.

(1) Use the CubeSuite+ code generator to generate the functions required to run the sample code. At this time, be sure to specify the following settings:

- Enable the overrun error callback feature. Select the check box for the overrun error in the callback feature settings for the 3-wire serial I/O (CSI21) that uses channel 1 of serial array unit 1.
- Set the output level of the reset pin to " H ".
  Enable the port to which the reset pin is connected and set the output level to High (1).

(2) Add r_sa_spi_control_register.c (and r_sa_spi_control_register.h) to the project.

(3) Add the following source code to the automatically generated file.

r_cg_serial_user.c file

```
//Include a header file.
#include "r_sa_spi_control_register.h"

//Declare a global variable for the overrun error flag.
volatile uint8_t   g_csi21_overrun_flag;

//Add the following processing to the r_csi21_callback_error function.

static void r_csi21_callback_error(uint8_t err_type)
{
    /* Start user code. Do not edit comment generated here */
        uint8_t                 dummy;

        /* Clear overrun flag of R5F10FMx register */
        dummy = SIO21;
        dummy = (uint8_t)(SSR11 & _0001_SAU_OVERRUN_ERROR);
        SIR11 = (uint16_t)dummy;

        /* Set overrun flag of global variable */
        g_csi21_overrun_flag = 1;

        /* Set CS output level high and stop CSI21 */
        SPI_CS = 1;
        R_CSI21_Stop();
    /* End user code. Do not edit comment generated here */
}

//Add the following processing to the r_csi21_callback_receiveend function.

static void r_csi21_callback_receiveend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Set CS output level high and stop CSI21 */
        SPI_CS = 1;
        R_CSI21_Stop();
    /* End user code. Do not edit comment generated here */
}
```

2. Conditions for Verifying Operation

The operation of the sample code shown in this application note has been verified under the conditions shown below.

Table 2.1 Conditions for Verifying Operation

Item	Description
Microcontroller used	RL78/G1E (R5F10FME)
Operating frequency	• High-speed on-chip oscillator (high-speed OCD) clock: 32 MHz • CPU/peripheral hardware clock: 32 MHz
Operating voltage	V_{DD}, DV_{DD}, AV_{DD1}, AV_{DD2}, AV_{DD3}: 5.0 V AV_{DD}: 3.3 V LVD detection voltage (V_{LVIH}): 4.06 V when rising, 3.98 V when falling
External devices used	None
Integrated development environment	CubeSuite+ V1.01.01 [31 Jan 2012] made by Renesas Electronics
C compiler (build tool)	CA78K0R V1.30 made by Renesas Electronics
RL78/G1A code library	CodeGenerator for RL78/G1A E1.00.00c [22 Dec 2011] made by Renesas Electronics

3. Related Application Notes

Related application notes are shown below. Also refer to these documents when using this application note.

- RL78/G13 Initialization (R01AN0451E) Application Note
- RL78/G13 Serial Array Unit for 3-Wire Serial I/O (Master Transmission/Reception) (R01AN0460E) Application Note

4. Functions

The functions used in the sample code described in this application note are listed in Table 4.1 below.

Table 4.1 Functions

Access	Function Name	Overview
Byte manipulation (8-bit access)	R_SPI_SmartAnalogRead	SPI control register read function
	R_SPI_SmartAnalogWrite	SPI control register write function
	R_SPI_SmartAnalogWriteVerify	SPI control register write verify function
Bit manipulation (1-bit access)	R_SPI_SmartAnalogReadBit	SPI control register bit read function
	R_SPI_SmartAnalogWriteBit	SPI control register bit write function
	R_SPI_SmartAnalogWriteVerifyBit	SPI control register bit write verify function

5. Return Values

The return values used in the sample code described in this application note are listed in Table 5.1 below.

Table 5.1 Return Values

Data Type	Return Values		
	Macro name	Value	Description
Spi_status_t (uint8_t)	SPI_OK	00H	Success
	SPI_ERR_PARAM	01H	Parameter error
	SPI_ERR_COM	02H	SPI communication error (overrun error, timeout error)
	SPI_ERR_VERIFY	03H	Verification error

6. Structures

The structures used in the sample code described in this application note are shown below.

6.1 Structures Used by Byte Manipulation Functions

The structures used by byte manipulation (8-bit access) functions are shown in Table 6.1 below.

Table 6.1 Structures Used by Byte Manipulation Functions

Structure data type name	`spi_data_t`		
Overview	Data format for reading and writing SPI control registers		
Data type size	2 bytes		
Member variables	Data type	Name	Description
	`uint8_t`	`address`	Address of the SPI control register
	`uint8_t`	`data`	Data in the SPI control register

6.2 Structures Used by Bit Manipulation Functions

The structures used by bit manipulation (1-bit access) functions are shown in Table 6.2 below.

Table 6.2 Structures Used by Bit Manipulation Functions

Structure data type name	`Spi_data_bit_t`		
Overview	Data format for bit-reading and bit-writing SPI control registers		
Data type size	3 bytes		
Member variables	Data type	Name	Description
	`uint8_t`	`address`	Address of the SPI control register
	`uint8_t`	`bitNum`	Number of bit in SPI control register (0 to 7)
	`uint8_t`	`bitData`	Setting of bit in SPI control register (0 or 1)

7. Function Specifications

The specifications of the functions used in the sample code described in this application note are described below.

7.1 SPI Control Register Read Function

The specifications of the SPI control register read function (R_SPI_SmartAnalogRead) are shown in Table 7.1 below.

Table 7.1 Specifications of SPI Control Register Read Function

Function name	R_SPI_ControlRegister_Read		
Overview	SPI control register read function		
Header file	r_sa_spi_control_register.h		
Declaration	spi_status_t R_SPI_SmartAnalogRead(spi_data_t *data, unit8_t num)		
Function description	Reads the data from an SPI control register in the analog block by using CSI21.		
Parameters	Data type	Name	Description
	spi_data_t *	data	Pointer to buffer storing the SPI control register address and read data bits
	uint8_t	num	Number of spi_data_t elements
Return value	Macro name	Value	Description
	SPI_OK	00H	Success
	SPI_ERR_COM	02H	SPI communication error (overrun error, timeout error)

7.2 SPI Control Register Write Function

The specifications of the SPI control register write function (R_SPI_SmartAnalogWrite) are shown in Table 7.2 below.

Table 7.2 Specifications of SPI Control Register Write Function

Function name	R_SPI_SmartAnalogWrite		
Overview	SPI control register write function		
Header file	r_sa_spi_control_register.h		
Declaration	spi_status_t R_SPI_SmartAnalogWrite(spi_data_t *data, unit8_t num)		
Function description	Writes data to an SPI control register in the analog block by using CSI21.		
Parameters	Data type	Name	Description
	spi_data_t *	data	Pointer to buffer storing the SPI control register address and data bits to be written
	uint8_t	num	Number of spi_data_t elements
Return value	Macro name	Value	Description
	SPI_OK	00H	Success
	SPI_ERR_COM	02H	SPI communication error (overrun error, timeout error)

7.3 SPI Control Register Write Verify Function

The specifications of the SPI control register write verify function (R_SmartAnalogWriteVerify) are shown in Table 7.3 below.

Table 7.3 Specifications of SPI Control Register Write Verify Function

Function name	`R_SPI_ControlRegisterWriteVerify`		
Overview	SPI control register write verify function		
Header file	`r_ra_spi_control_register.h`		
Declaration	`spi_status_t R_SPI_SmartAnalogWriteVerify(spi_data_t *data, unit8_t num, uint8_t *errIndex)`		
Function description	Writes data to an SPI control register in the analog block by using CSI21, and then verifies that the data has been written.		
Parameters	Data type	Name	Description
	`spi_data_t *`	`data`	Pointer to buffer storing the SPI control register address and data bits to be written
	`uint8_t`	`num`	Number of spi_data_t elements
	`uint8_t *`	`errIndex`	Pointer to buffer storing the number of the spi_data_t element that caused the verify error (0 to *num* – 1)
Return value	Macro name	Value	Description
	`SPI_OK`	00H	Success
	`SPI_ERR_COM`	02H	SPI communication error (overrun error, timeout error)
	`SPI_ERR_VERIFY`	03H	Verification error

7.4 SPI Control Register Bit Read Function

The specifications of the SPI control register bit read function (R_SPI_SmartAnalogReadBit) are shown in Table 7.4 below.

Table 7.4 Specifications of SPI Control Register Bit Read Function

Function name	`R_SPI_SmartAnalogReadBit`		
Overview	SPI control register bit read function		
Header file	`r_sa_spi_control_register.h`		
Declaration	`Spi_status_t R_SPI_SmartAnalogReadBit(SPI_DATA_BIT *data, unit8_t num, uint8_t *errIndex)`		
Function description	Reads the bit data from an SPI control register in the analog block by using CSI21.		
Parameters	Data type	Name	Description
	`spi_data_bit_t *`	`data`	Pointer to buffer storing the SPI control register address, bit number, and read data bit
	`uint8_t`	`num`	Number of spi_data_bit_t elements
	`uint8_t *`	`errIndex`	Pointer to buffer storing the number of the spi_data_t element that caused the parameter error (0 to *num* – 1)
Return value	Macro name	Value	Description
	`SPI_OK`	00H	Success
	`SPI_ERR_PARAM`	01H	Parameter error (illegal bit)
	`SPI_ERR_COM`	02H	SPI communication error (overrun error, timeout error)

7.5 SPI Control Register Bit Write Function

The specifications of the SPI control register bit write function (R_SPI_SmartAnalogWriteBit) are shown in Table 7.5 below.

Table 7.5 Specifications of SPI Control Register Bit Write Function

Function name	`R_SPI_SmartAnalogWriteBit`		
Overview	SPI control register bit write function		
Header file	`SPI_ControlRegister.h`		
Declaration	`SPI_STATUS SPI_ControlRegister_Write_Bit(SPI_DATA_BIT *data, unit8_t num, uint8_t *errIndex)`		
Function description	Writes bit data to an SPI control register in the analog block by using CSI21.		
Parameters	Data type	Name	Description
	`spi_data_bit_t *`	`data`	Pointer to buffer storing the SPI control register address, bit number, and write data bit
	`uint8_t`	`num`	Number of spi_data_bit_t elements
	`uint8_t *`	`errIndex`	Pointer to buffer storing the number of the spi_data_bit_t element that caused the parameter error (0 to *num* – 1)
Return value	Macro name	Value	Description
	`SPI_OK`	00H	Success
	`SPI_ERR_PARAM`	01H	Parameter error (invalid bit number or bit data)
	`SPI_ERR_COM`	02H	SPI communication error (overrun error, timeout error)

7.6 SPI Control Register Bit Write Verify Function

The specifications of the SPI control register bit write verify function (R_SPI_SmartAnalogWriteVerifyBit) are shown in Table 7.6 below.

Table 7.6 Specifications of SPI Control Register Bit Write Verify Function

Function name	`R_SPI_SmartAnalogWriteVerifyBit`		
Overview	SPI control register bit write verify function		
Header file	`r_sa_spi_control_register.h`		
Declaration	`spi_status_t R_SPI_SmartAnalogWriteVerifyBit(spi_data_bit_t *data, unit8_t num, uint8_t *errIndex)`		
Function description	Writes bit data to an SPI control register in the analog block by using CSI21, and then verifies that the data has been written.		
Parameters	Data type	Name	Description
	`spi_data_bit_t *`	`data`	Pointer to buffer storing the SPI control register address, bit number, and write data bit
	`uint8_t`	`num`	Number of spi_data_bit_t elements
	`uint8_t *`	`errIndex`	Pointer to buffer storing the number of the spi_data_bit_t element that caused the parameter error or verification error (0 to *num* – 1)
Return value	Macro name	Value	Description
	`SPI_OK`	00H	Success
	`SPI_ERR_PARAM`	01H	Parameter error (invalid bit number or bit data)
	`SPI_ERR_COM`	02H	SPI communication error (overrun error, timeout error)
	`SPI_ERR_VERIFY`	03H	Verification error

8. Flowcharts

The flowcharts for the sample code described in this application note are shown below.

8.1 SPI Control Register Read Function

The flowchart for the SPI control register read function is shown in Figure 8.1 below.

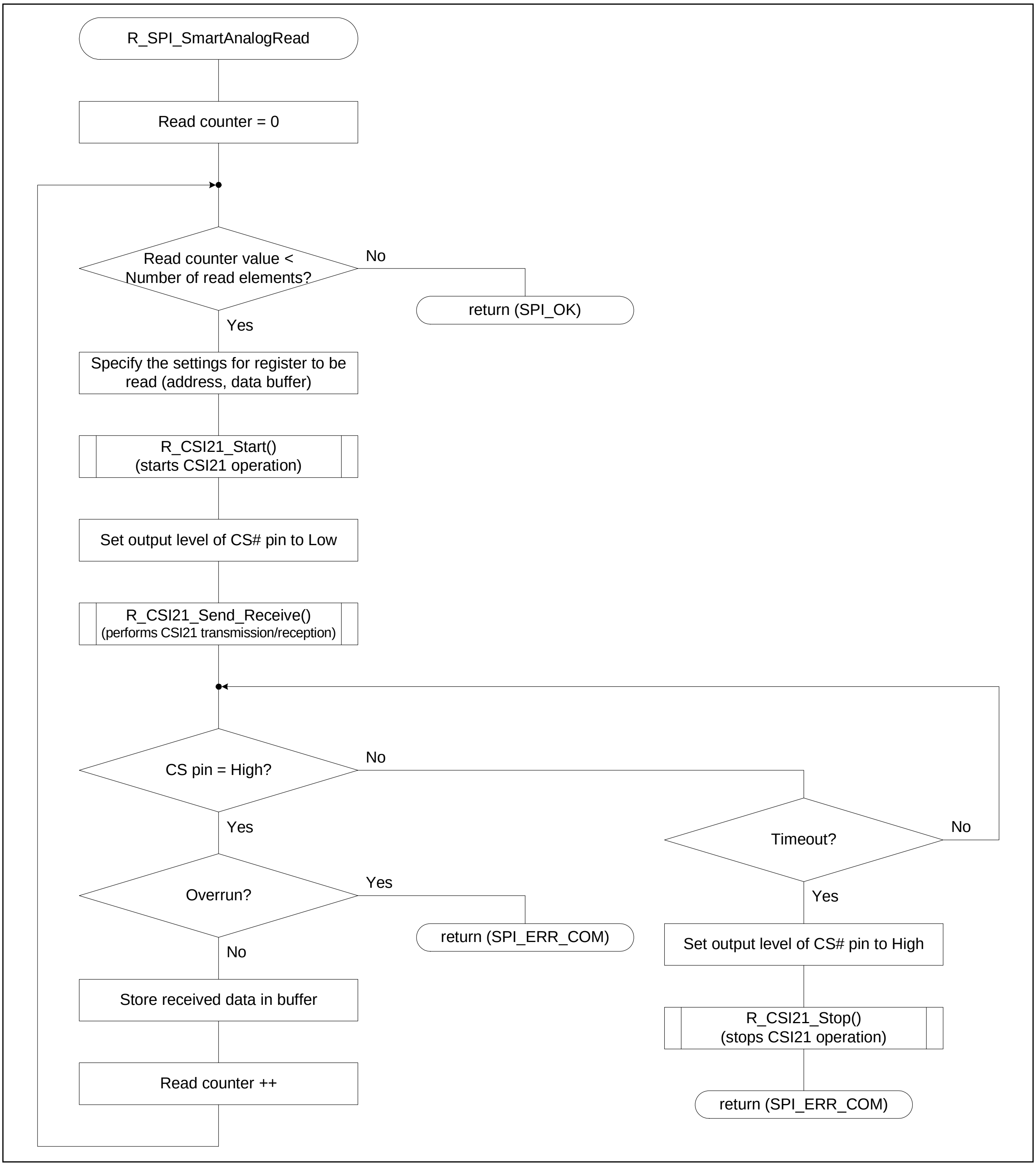
R_SPI_SmartAnalogRead
Read counter = 0
Read counter value < Number of read elements?
No
return (SPI_OK)
Yes
Specify the settings for register to be read (address, data buffer)
R_CSI21_Start() (starts CSI21 operation)
Set output level of CS# pin to Low
R_CSI21_Send_Receive() (performs CSI21 transmission/reception)
CS pin = High?
No
Timeout?
No
Yes
Set output level of CS# pin to High
R_CSI21_Stop() (stops CSI21 operation)
return (SPI_ERR_COM)
Yes
Overrun?
Yes
return (SPI_ERR_COM)
No
Store received data in buffer
Read counter ++

Figure 8.1 Flowchart for SPI Control Register Read Function

8.2 SPI Control Register Write Function

The flowchart for the SPI control register write function is shown in Figure 8.2 below.

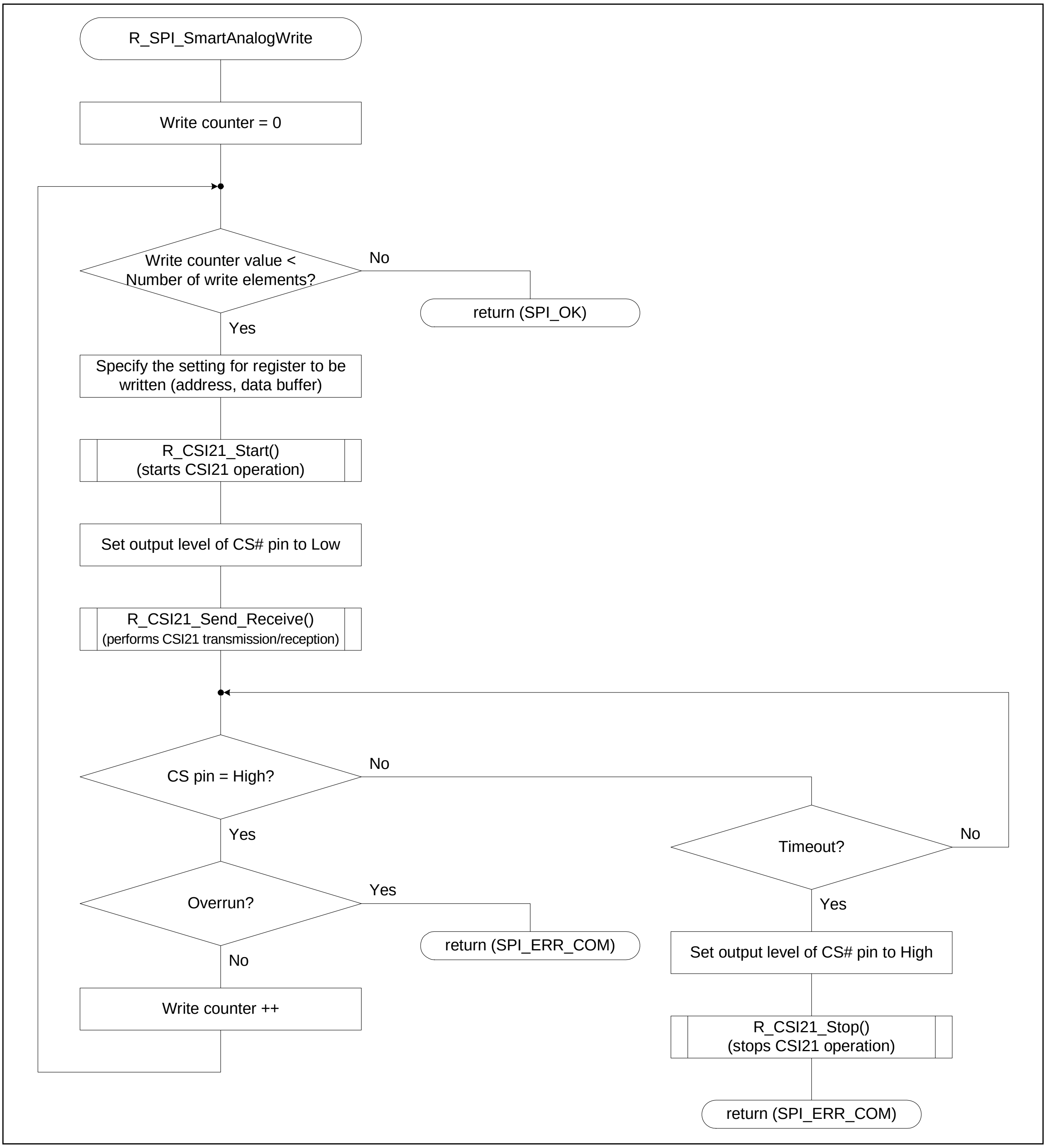
R_SPI_SmartAnalogWrite
Write counter = 0
Write counter value < Number of write elements?
No
return (SPI_OK)
Yes
Specify the setting for register to be written (address, data buffer)
R_CSI21_Start() (starts CSI21 operation)
Set output level of CS# pin to Low
R_CSI21_Send_Receive() (performs CSI21 transmission/reception)
CS pin = High?
No
Yes
Timeout?
No
Yes
Overrun?
Yes
return (SPI_ERR_COM)
No
Write counter ++
Set output level of CS# pin to High
R_CSI21_Stop() (stops CSI21 operation)
return (SPI_ERR_COM)

Figure 8.2 Flowchart for SPI Control Register Write Function

8.3 SPI Control Register Write Verify Function

The flowchart for the SPI control register write verify function is shown in Figure 8.3 below.

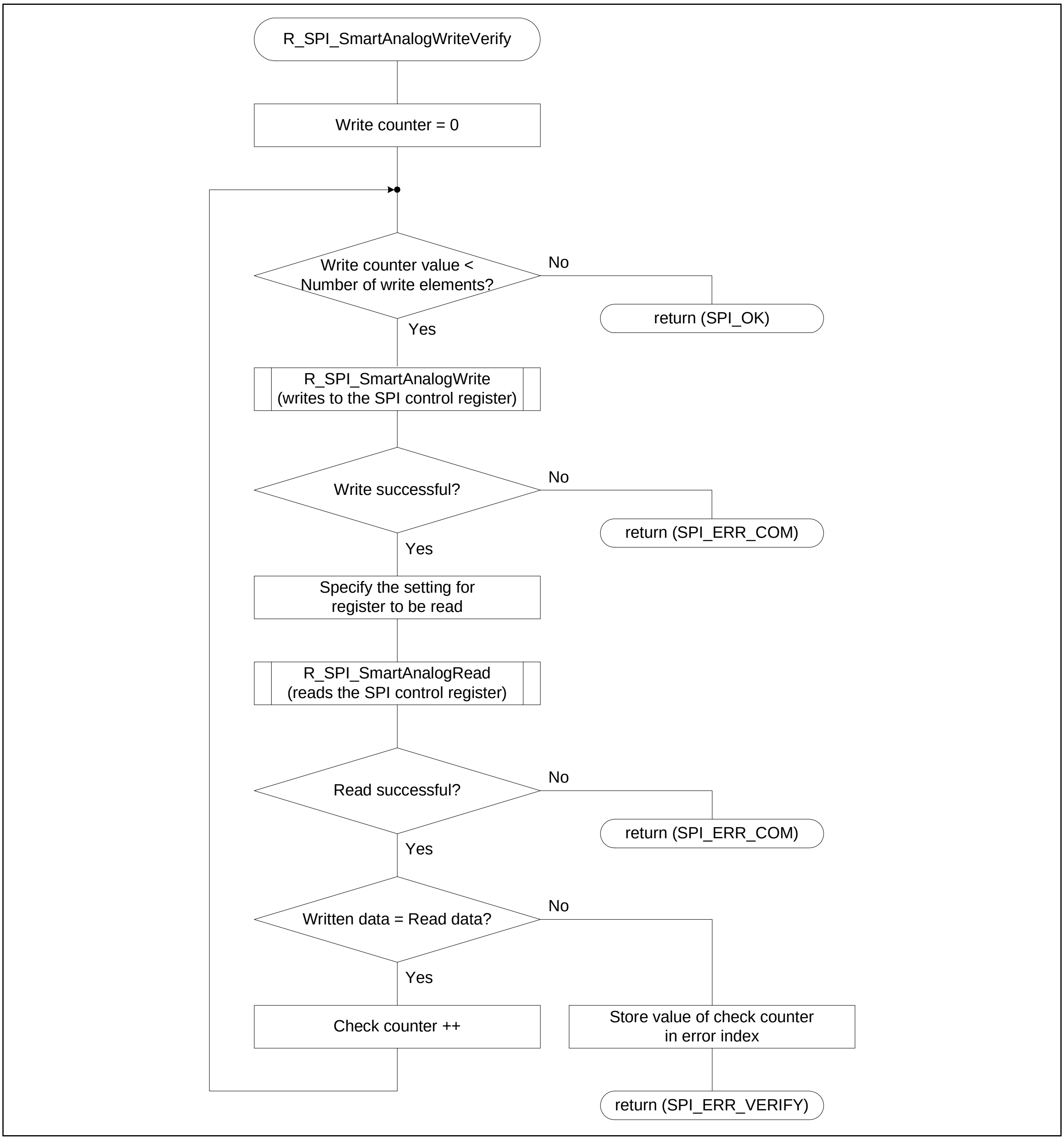
R_SPI_SmartAnalogWriteVerify
Write counter = 0
Write counter value < Number of write elements?
No
return (SPI_OK)
Yes
R_SPI_SmartAnalogWrite (writes to the SPI control register)
Write successful?
No
return (SPI_ERR_COM)
Yes
Specify the setting for register to be read
R_SPI_SmartAnalogRead (reads the SPI control register)
Read successful?
No
return (SPI_ERR_COM)
Yes
Written data = Read data?
No
Store value of check counter in error index
return (SPI_ERR_VERIFY)
Yes
Check counter ++

Figure 8.3 Flowchart for SPI Control Register Write Verify Function

8.4 SPI Control Register Bit Read Function

The flowchart for the SPI control register bit read function is shown in Figure 8.4 below.

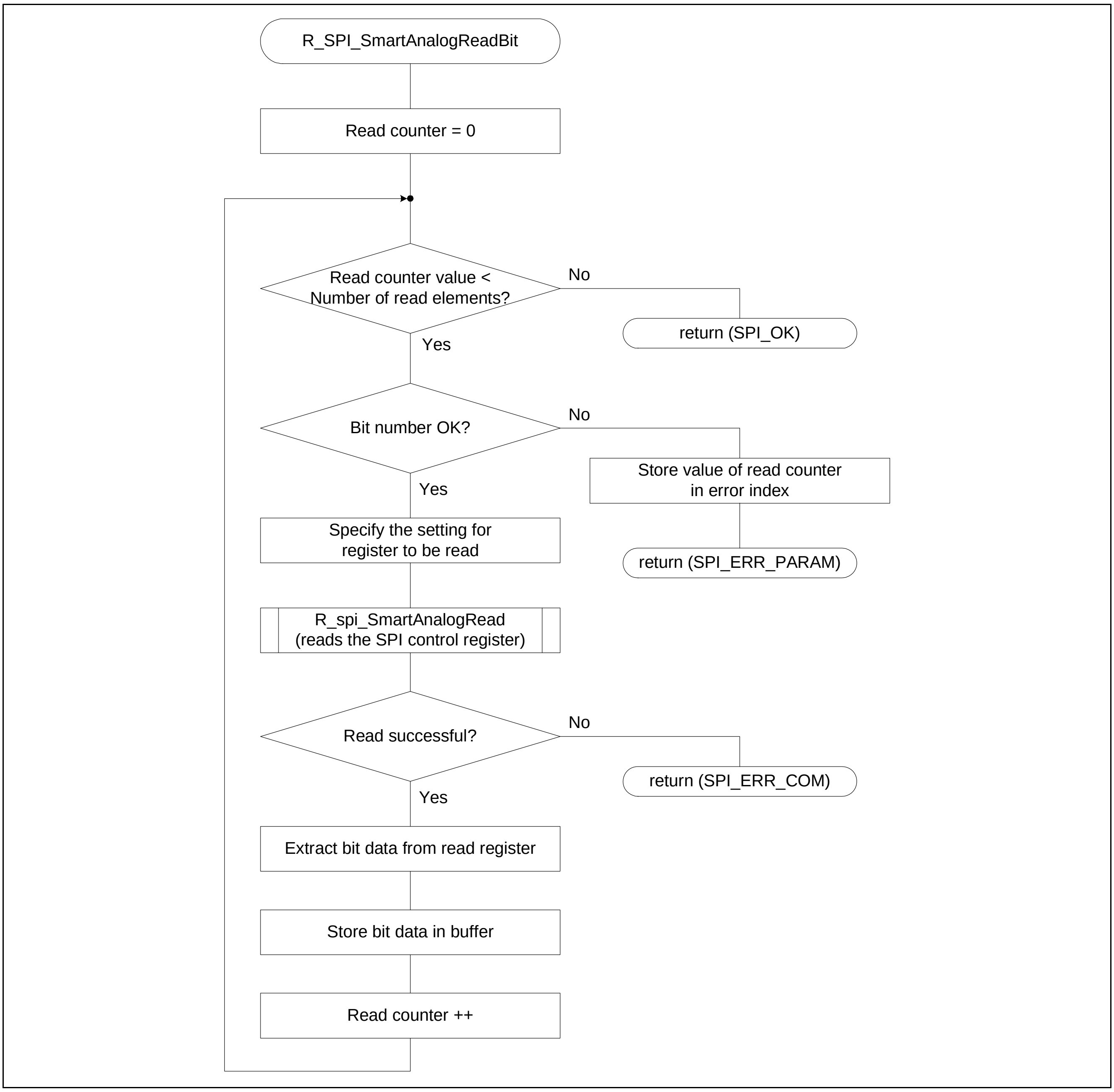
R_SPI_SmartAnalogReadBit
Read counter = 0
Read counter value < Number of read elements?
No
return (SPI_OK)
Yes
Bit number OK?
No
Store value of read counter in error index
return (SPI_ERR_PARAM)
Yes
Specify the setting for register to be read
R_spi_SmartAnalogRead (reads the SPI control register)
Read successful?
No
return (SPI_ERR_COM)
Yes
Extract bit data from read register
Store bit data in buffer
Read counter ++

Figure 8.4 Flowchart for SPI Control Register Bit Read Function

8.5 SPI Control Register Bit Write Function

The flowchart for the SPI control register bit write function is shown in Figure 8.5 below.

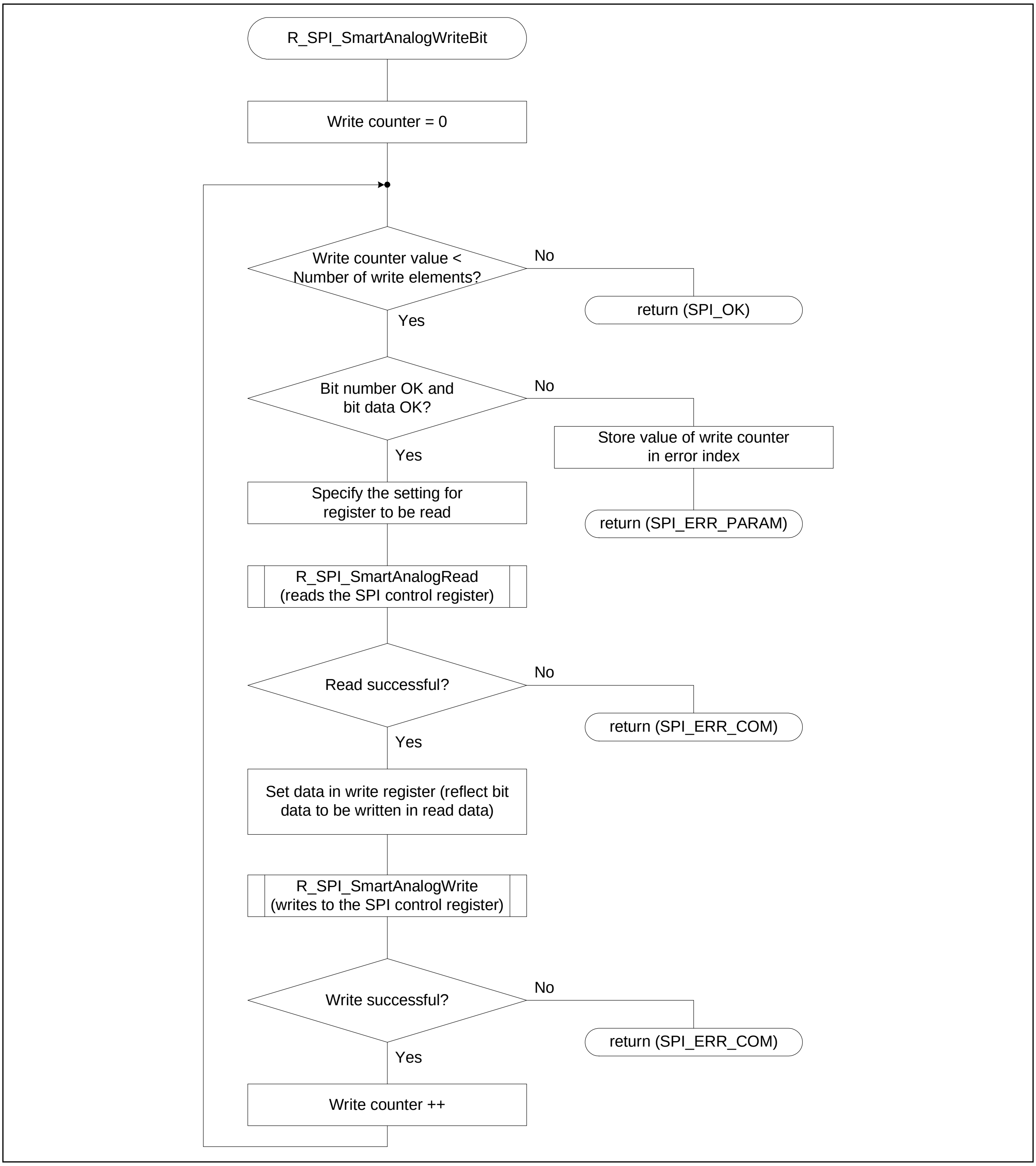
R_SPI_SmartAnalogWriteBit
Write counter = 0
Write counter value < Number of write elements?
No
return (SPI_OK)
Yes
Bit number OK and bit data OK?
No
Store value of write counter in error index
return (SPI_ERR_PARAM)
Yes
Specify the setting for register to be read
R_SPI_SmartAnalogRead (reads the SPI control register)
Read successful?
No
return (SPI_ERR_COM)
Yes
Set data in write register (reflect bit data to be written in read data)
R_SPI_SmartAnalogWrite (writes to the SPI control register)
Write successful?
No
return (SPI_ERR_COM)
Yes
Write counter ++

Figure 8.5 Flowchart for SPI Control Register Bit Write Function

8.6 SPI Control Register Bit Write Verify Function

The flowchart for the SPI control register bit write verify function is shown in Figure 8.6 below.

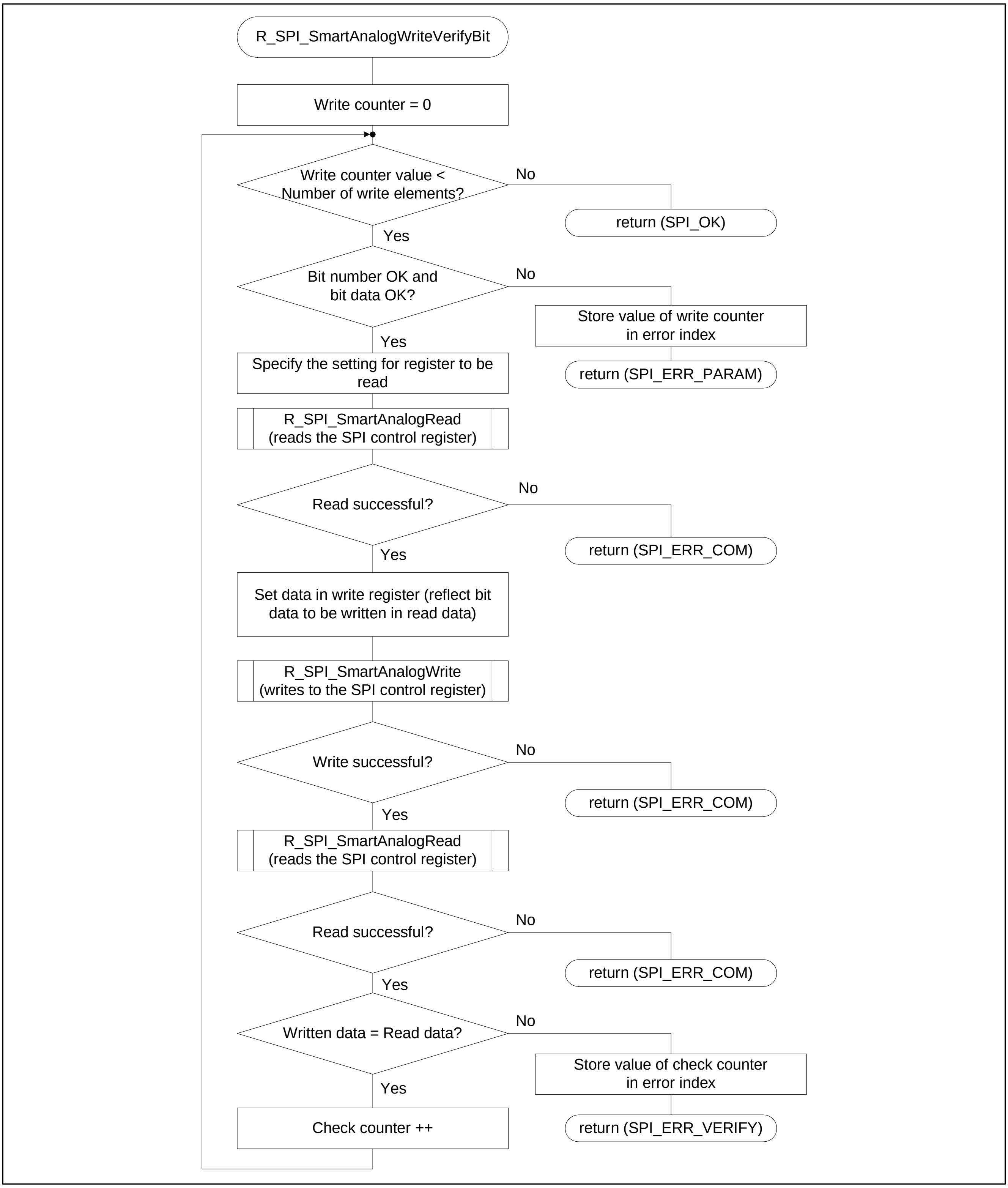
R_SPI_SmartAnalogWriteVerifyBit
Write counter = 0
Write counter value < Number of write elements?
No
return (SPI_OK)
Yes
Bit number OK and bit data OK?
No
Store value of write counter in error index
return (SPI_ERR_PARAM)
Yes
Specify the setting for register to be read
R_SPI_SmartAnalogRead (reads the SPI control register)
Read successful?
No
return (SPI_ERR_COM)
Yes
Set data in write register (reflect bit data to be written in read data)
R_SPI_SmartAnalogWrite (writes to the SPI control register)
Write successful?
No
return (SPI_ERR_COM)
Yes
R_SPI_SmartAnalogRead (reads the SPI control register)
Read successful?
No
return (SPI_ERR_COM)
Yes
Written data = Read data?
No
Store value of check counter in error index
return (SPI_ERR_VERIFY)
Yes
Check counter ++

Figure 8.6 Flowchart for SPI Control Register Bit Write Verify Function

9. Examples of Using the Sample Code

Examples of using the sample code described in this application note are described below.

9.1 Example of Using SPI Control Register Read Function

An example of using the SPI control register read function (R_SPI_SmartAnalogRead) is shown in Figure 9.1 below.

■ Example of use

(1) Address 00H is read.
(2) Address 01H is read.
(3) Address 03H is read.
(4) Address 04H is read.
(5) Address 05H is read.

```
#include "r_cg_macrodriver.h"
#include "r_sa_control_register.h"

void main(void)
{
        uint8_t               errCode;     // For storing the function's return value
        uint8_t               temp[5];

        // Prepare a buffer for read data and specify the address
        // (Any value can be specified as the initial value of the data when read)
        spi_data_t            readData[5] = {
               {0x00, 0x00},                // address: 00H, data: 00H(dummy)
               {0x01, 0x00},                // address: 01H, data: 00H(dummy)
               {0x03, 0x00},                // address: 03H, data: 00H(dummy)
               {0x04, 0x00},                // address: 04H, data: 00H(dummy)
               {0x05, 0x00}                 // address: 05H, data: 00H(dummy)
        };

        // Read the SPI control register
        errCode = R_SPI_SmartAnalogRead(readData, 5);

        // Error check
        if (errCode != SPI_OK) {
               // Error handling
        }
        else {
               // Obtain read data
               temp[0] = readData[0].data;            // Obtain data at address 00H
               temp[1] = readData[1].data;            // Obtain data at address 01H
               temp[2] = readData[2].data;            // Obtain data at address 03H
               temp[3] = readData[3].data;            // Obtain data at address 04H
               temp[4] = readData[4].data;            // Obtain data at address 05H
        }
        while(1);
}
```

Figure 9.1 Example of Using SPI Control Register Read Function

9.2 Example of Using SPI Control Register Write Function

An example of using the SPI control register write function (R_SPI_SmartAnalogWrite) is shown in Figure 9.2 below.

- Example of use
  (1) Data 57H is written to address 11H.

```
#include "r_cg_macrodriver.h"
#include "r_sa_spi_control_register.h"

void main(void)
{
        uint8_t                 errCode;    // For storing the function's return value

        // Prepare a buffer for write data and specify the address
        spi_data_t              writeData[1] = {
                {0x11, 0x57}                // address: 11H, data: 57H
        };

        // Write to the SPI control register
        errCode = R_SPI_SmartAnalogWrite(writeData, 1);

        // Error check
        if (errCode != SPI_OK) {
                // Error handling
        }
        else {
                // Normal processing
        }

        while(1);
}
```

Figure 9.2 Example of Using SPI Control Register Write Function

9.3 Example of Using SPI Control Register Write Verify Function

An example of using the SPI control register write verify function (R_SPI_SmartAnalogWriteVerify) is shown in Figure 9.3 below.

- Example of use
  (4) Data 0DH is written to address 0BH.
  (5) Data 02H is written to address 12H.
  (6) After (1) and (2) are executed, verify processing is executed.

```
#include "r_cg_macrodriver.h"
#include "r_sa_spi_control_register.h"

void main(void)
{
        uint8_t         errCode;                // For storing the function's return value
        uint8_t         errIndex;   // For storing error index
        uint8_t         temp;

        // Prepare a buffer for write data and specify the address
        spi_data_t      writeData[2] = {
                {0x0B, 0x0D},           // address: 0BH, data: 0DH
                {0x12, 0x02}            // address: 12H, data: 02H
        };

        // Verify that data has been written to the SPI control register
        errCode = R_SPI_SmartAnalogWriteVerify(writeData, 2, &errIndex);

        // Error check
        if (errCode == SPI_OK) {
                // Normal processing
        }
        else if (errCode == SPI_ERR_VERIFY){
                // If a verification error occurs
                temp = errIndex;        // Obtain the index of the cause of the verification error
        }
        else {
                // Other error handling
        }

        while(1);
}
```

Figure 9.3 Example of Using SPI Control Register Write Verify Function

9.4 Example of Using SPI Control Register Bit Read Function

An example of using the SPI control register bit read function (R_SPI_SmartAnalogReadBit) is shown in Figure 9.4 below.

- Example of use
  (1) Bit 6 of address 01H is read.

```
#include "r_cg_macrodriver.h"
#include "r_sa_spi_control_register.h"

void main(void)
{
        uint8_t             errCode;                    // For storing the function's return value
        uint8_t             errIndex;    // For storing error index
        uint8_t             temp;

        // Prepare a buffer for read data and specify the address and bit number
        // (Any value can be specified as the initial value of the bit data when read)
        spi_data_bit_t      readData[1] = {
               {0x01, 6, 0}                             // address: 01H, bitNum: 6, bitData: 0(dummy)
        };

        // Read the specified bit of the SPI control register
        errCode = R_SPI_SmartAnalogReadBit(readData, 1, &errIndex);

        // Error check
        if (errCode == SPI_OK) {
               // Obtain the read bit data
               temp = readData[0].bitData;              // Obtain the data of bit 6 of address 01H
        }
        else if (errCode == SPI_ERR_PARAM){
               // If a parameter error occurs
               temp = errIndex;                                        // Obtain the index of the cause of the parameter error
        }
        else {
               // Other error handling
        }

        while(1);
}
```

Figure 9.4 Example of Using SPI Control Register Bit Read Function

9.5 Example of Using SPI Control Register Bit Write Function

An example of using the SPI control register bit write function (R_SPI_SmartAnalogWriteBit) is shown in Figure 9.5 below.

- Example of use
  (1) 1 is written to bit 2 of address 11H.
  (2) 0 is written to bit 4 of address 12H.

```
#include "r_cg_macrodriver.h"
#include "r_sa_spi_control_register.h"

void main(void)
{
        uint8_t             errCode;                // For storing the function's return value
        uint8_t             errIndex;    // For storing error index
        uint8_t             temp;

        // Prepare a buffer for write data and specify the address, bit number, and bit data
        spi_data_bit_t      writeData[2] = {
                {0x11, 2, 1},                       // address: 11H, bitNum: 2, bitData: 1
                {0x12, 4, 0}            // address: 12H, bitNum: 4, bitData: 0
        };

        // Write to the specified bit of the SPI control register
        errCode = R_SPI_SmartAnalogWriteBit(writeData, 2, &errIndex);

        // Error check
        if (errCode == SPI_OK) {
                // Normal processing
        }
        else if (errCode == SPI_ERR_PARAM){
                // If a parameter error occurs
                temp = errIndex;        // Obtain the index of the cause of the parameter error
        }
        else {
                // Other error handling
        }

        while(1);
}
```

Figure 9.5 Example of Using SPI Control Register Bit Write Function

9.6 Example of Using SPI Control Register Bit Write Verify Function

An example of using the SPI control register bit write verify function (`SPI_ControlRegister_Write_Verify_Bit`) is shown in Figure 9.6 below.

- Example of use
  (1) 1 is written to bit 1 of address 01H.
  (2) 0 is written to bit 2 of address 11H.
  (3) 1 is written to bit 3 of address 12H.
  (4) After (1), (2) and (3) are executed, verify processing is executed.

```
#include "r_cg_macrodriver.h"
#include "r_sa_spi_control_register.h"

void main(void)
{
        uint8_t             errCode;                    // For storing the function's return value
        uint8_t             errIndex;    // For storing error index
        uint8_t             temp;

        // Prepare a buffer for write data and specify the address, bit number, and bit data
        spi_data_bit_t      writeData[3] = {
                {0x01, 1, 1},                           // address: 01H, bitNum: 1, bitData: 1
                {0x11, 2, 0},                           // address: 11H, bitNum: 2, bitData: 0
                {0x12, 3, 1}                 // address: 12H, bitNum: 3, bitData: 1
        };

        // Verify that the specified bit of the SPI control register has been written
        errCode = R_SPI_SmartAnalogWriteVerifyBit(writeData, 3, &errIndex);

        // Error check
        if (errCode == SPI_OK) {
                // Normal processing
        }
        else if (errCode == SPI_ERR_PARAM){
                // If a parameter error occurs
                temp = errIndex;            // Obtain the index of the cause of the parameter error
        }
        else if (errCode == SPI_ERR_VERIFY){
                // If a verification error occurs
                temp = errIndex;            // Obtain the index of the cause of the verification error
        }
        else {
                // Other error handling
        }

        while(1);
}
```

Figure 9.6 Example of Using SPI Control Register Bit Write Verify Function

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep. 30, 2012	—	First edition issued.
1.10	Mar. 29, 2013	—	Change of descriptions
1.20	Sep. 30, 2013	—	Addition of 1.2 Procedure for using the sample code

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 manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   - Handle unused pins in accord 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 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 one with a different part number, confirm that the change will not lead to problems.

   — The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141