RZ/N2L Group

SSI Sample Program

Introduction

This application note explains a sample program for acquiring and indicating information from an encoder compliant with synchronized serial interface (“SSI”) specification by using serial peripheral interface (SPI) of the RZ/N2L.

The major features of the program are listed below.

- Supports encoders compliant with SSI specification.
- Reads out and indicates rotation angles, etc. from positional encoders.

Target Device

RZ/N2L

Contents

1. Specifications

Table 1.1 lists the peripheral modules to be used and their applications. Figure 1.1 shows the operating environment when the sample code is being executed.

Table 1.1 Peripheral Modules and Applications

Peripheral Module	Application
Serial peripheral interface (SPI)	Communicates with the SSI compliant encoder.
Interrupt controller (ICU)	Controls interrupts for SSI communication controller.
General PWM timer (GPT)	Generates communication intervals for continuous mode and generates event intervals for activating DMAC.
DMA Controller (DMAC)	Activates by the event from GPT unit 0 channel 1 and generates TX trigger synchronized with the event.
Serial communication interface (SCI) UART	Asynchronous communications of the SCI are used for COM port communications by using USB interface. It is used for console interface of the sample program.

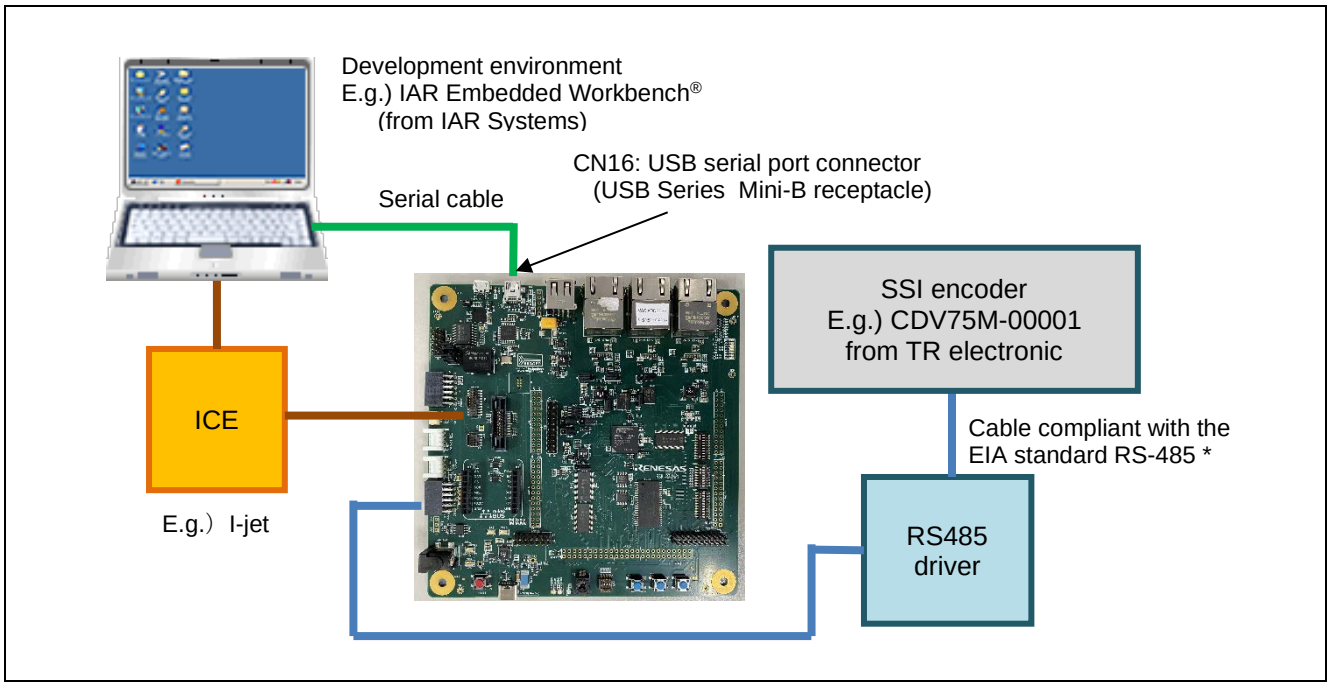


Figure 1.1 Operating Environment

Note: For allowable cable length for transmission and reception, contact the manufacturer of the encoder.

2. Operating Environment

The sample code covered in this application note is for the environment below.

Table 2.1 Operating Environment

Item	Description
MCU	RZ/N2L Group
Operating frequency	CPUCLK = 400 MHz
Operating voltage	1.1 V (Core) / 1.8 V (PLL, etc.) / 3.3 V (I/O)
Integrated development environment *	IAR Systems: IAR Embedded Workbench® for Arm® RENESAS: e² studio
Board	RSK+RZN2L (RTK9RZN2L0C00000BE)
Devices (function to be used on the board)	None

Note. Refer to the RZ/N2L Group Encoder I/F SSI sample program Release Note to check the version information of each integrated development environment.

3. Peripheral Functions

The basics of the peripheral modules, operating modes, and registers are described in the “RZ/N2L Group User’s Manual: Hardware”.

3.1 Pins

Table 3.1 lists the pins used and their functions.

Table 3.1 Pins Used and Their Functions

Channel	Port Name	I/O Port	Input/Output	Description
SSI0	SPI_MISO1	P14_7	Input	Data input
	SPI_RSPCK1	P04_4	Output	Clock output
SSI1	SPI_MISO2	P18_6	Input	Data input
	SPI_RSPCK2	P18_4	Output	Clock output

4. Software

4.1 SSI Driver Function

The functions of the SSI driver are listed below.

1. Initial settings
2. Acquisition of reception data
3. Notification of errors in transfer from the encoder

4.2 File Structures

For the file structure, refer to the release note for the RZ/N2L Group Encoder I/F SSI sample program.

4.3 Functions

Table 4.1 lists the functions to be used.

Table 4.1 List of Functions

Category	Function Name	Page Number
SSI I/F driver API functions	R_SSI_Open	8
	R_SSI_Close	8
	R_SSI_GetVersion	8
	R_SSI_Control	9
	R_SSI_GetStatus	12
	R_SSI_GetRxd	12
User-defined functions	ssi_int_ssi_callback	13
	ssi_int_fifo_callback	13
Interrupt handlers	enc_ch0_rxi_isr	13
	enc_ch0_eri_isr	14
	enc_ch1_rxi_isr	14
	enc_ch1_eri_isr	14

4.4 Specification of API Functions

4.4.1 R_SSI_Open

R_SSI_Open	
Synopsis	Starts controlling operation of the encoder.
Header	r_ssi_rzt2_if.h
Declaration	int32_t R_SSI_Open(const int32_t id);
Description	This function is used for the following initial settings of the SSI communication channel. 1. Initial setting of the serial peripheral interface (SPI) 2. Initializing FIFO
Arguments	id : Designates the ID to be used. (It is defined in r_ssi_rzt2_dat.h.) R_SSI0_ID : Specifies channel 0. R_SSI1_ID : Specifies channel 1. Others : Setting is not allowed.
Return value	R_SSI_SUCCESS : Normal termination R_SSI_ERR_ACCESS : Abnormal termination (The channel is already open.) R_SSI_ERR_INVALID_ARG : Abnormal termination (The id is not specified.)
Note	Calling this API function from within a callback function is not allowed.

4.4.2 R_SSI_Close

R_SSI_Close	
Synopsis	Stops controlling operation of the encoder.
Header	r_ssi_rzt2_if.h
Declaration	int32_t R_SSI_Close(const int32_t id);
Description	This function stops controlling operation of the encoder on the designated channel.
Arguments	id : Designates the ID to be used. (It is defined in r_ssi_rzt2_dat.h.) R_SSI0_ID : Specifies channel 0. R_SSI1_ID : Specifies channel 1. Others : Setting is not allowed.
Return value	R_SSI_SUCCESS : Normal termination R_SSI_ERR_ACCESS : Abnormal termination (Transfer is in progress.) R_SSI_ERR_INVALID_ARG : Abnormal termination (The id is not specified.)
Note	If the transfer is in progress, the operation cannot be closed. Calling this API function from within a callback function is not allowed.

4.4.3 R_SSI_GetVersion

R_SSI_GetVersion	
Synopsis	Acquires the version number of the encoder interface driver.
Header	r_ssi_rzt2_if.h
Declaration	uint32_t R_SSI_GetVersion();
Description	This function acquires the version number of the SSI driver.
Arguments	None
Return value	A major part of the version number is stored in the 16 MSBs and the minor part of the version number is stored in the 16 LSBs. Ex.) For ver.1.2, the value returned is 0x00010002.

4.4.4 R_SSI_Control

R_SSI_Control		
Synopsis	Controls operation of the encoder.	
Header	r_ssi_rzt2_if.h	
Declaration	int32_t R_SSI_Control(const int32_t id, const r_ssi_cmd_t cmd, const void *p_buf);	
Description	This function controls the operations of the encoder by using the cmd argument.	
Arguments	id	: Designates the ID to be used. (It is defined in r_ssi_rzt2_dat.h.)
		R_SSI0_ID : Specifies channel 0.
		R_SSI1_ID : Specifies channel 1.
		Others : Setting is not allowed.
	cmd	: Command For the list of commands, see “Table 4.10 r_ssi_cmd_t”.
	p_buf	: Arguments corresponding to each command
Return value	R_SSI_SUCCESS	: Normal termination
	R_SSI_ERR_ACCESS	: Abnormal termination (The channel is not open.)
	R_SSI_ERR_INVALID_ARG	: Abnormal termination (The id or cmd is invalid.)
	See “4.4.4(1) Control Commands” for other returned values.	
Note	Be sure to call R_SSI_Open before calling this function.	

(1) Control Commands

(a) R_SSI_CMD_SET_PARAM

R_SSI_CMD_SET_PARAM		
Synopsis	Sets SSI communication parameters.	
Header	r_ssi_rzt2_if.h	
Declaration	int32_t R_SSI_Control(const int32_t id, const r_ssi_cmd_t cmd, const void *p_buf);	
Description	This function sets SSI communication parameters designated by arguments to the SSI communication channel. This function sets SPBR, SPSCR, SPCMD, SPDCR, SPDCR2, SPCR registers of the SPI.	
Arguments	id	: Designates the ID to be used. (It is defined in r_ssi_rzt2_dat.h.)
		R_SSI0_ID : Specifies channel 0.
		R_SSI1_ID : Specifies channel 1.
		Others : Setting is not allowed.
	cmd	: Designates R_SSI_CMD_SET_PARAM.
	p_buf	: Communication parameter Designates pointer to the structure r_ssi_param_t where the communication parameter is stored. See “4.10.1(1) r_ssi_param_t” for details.
Return value	R_SSI_SUCCESS	: Normal termination
	R_SSI_ERR_ACCESS	: Abnormal termination (The channel id not open.)
	R_SSI_ERR_INVALID_ARG	: Abnormal termination (The id or cmd is invalid, p_buf is NULL, or the structure r_ssi_param_t member is not specified.)
	R_SSI_ERR_BUSY	: Abnormal termination (The data transfer or the timer event synchronized process is in progress.)
Note	Be sure to call R_SSI_Open before calling this function. Calling this API function from within a callback function is not allowed.	

(b) R_SSI_CMD_SET_CALLBACK

R_SSI_CMD_SET_CALLBACK		
Synopsis	Sets callback functions of the encoder controller.	
Header	r_ssi_rzt2_if.h	
Declaration	int32_t R_SSI_Control(const int32_t id, const r_ssi_cmd_t cmd, const void *p_buf);	
Description	This function sets callback functions of the SSI communication channel.	
Arguments	id	: Designates the ID to be used. (It is defined in r_ssi_rzt2_dat.h.)
		R_SSI0_ID : Specifies channel 0.
		R_SSI1_ID : Specifies channel 1.
		Others : Setting is not allowed.
	cmd	: Designates R_SSI_CMD_SET_CALLBACK.
	p_buf	: Callback function information Designates pointer to the structure r_ssi_callback_t where the callback function information is stored. See “4.10.1(5) r_ssi_callback_t” for details. If NULL is designated as p_buf, callback function is not registered.
Return value	R_SSI_SUCCESS	: Normal termination
	R_SSI_ERR_ACCESS	: Abnormal termination (The channel is not open.)
	R_SSI_ERR_INVALID_ARG	: Abnormal termination (The id or cmd is invalid.)
	R_SSI_ERR_BUSY	: Abnormal termination (The data transfer or the timer event synchronized process is in progress.)
Note	Be sure to call R_SSI_Open before calling this function. Calling this API function from within a callback function is not allowed.	

(c) R_SSI_CMD_START

R_SSI_CMD_START		
Synopsis	Starts SSI communication.	
Header	r_ssi_rzt2_if.h	
Declaration	int32_t R_SSI_Control(const int32_t id, const r_ssi_cmd_t cmd, const void *p_buf);	
Description	This function starts the SSI communication. In response to the communication, callback function registered by the R_SSI_CMD_SET_CALLBACK command is called. See “4.5.1 ssi_int_ssi_callback” and “4.5.2 ssi_int_fifo_callback” for details of the callback functions.	
Arguments	id	: Designates the ID to be used. (It is defined in r_ssi_rzt2_dat.h.)
		R_SSI0_ID : Specifies channel 0.
		R_SSI1_ID : Specifies channel 1.
		Others : Setting is not allowed.
	cmd	: Designates R_SSI_CMD_START.
	p_buf	: Not used. (Designate NULL.)
Return value	R_SSI_SUCCESS	: Normal termination
	R_SSI_ERR_ACCESS	: Abnormal termination (The channel is not open.)
	R_SSI_ERR_INVALID_ARG	: Abnormal termination (The id or cmd is invalid.)
	R_SSI_ERR_BUSY	: Abnormal termination (The data transfer or the timer event synchronized process is in progress.)
Note	Be sure to call R_SSI_Open before calling this function.	

(d) R_SSI_CMD_ELC_START

R_SSI_CMD_ELC_START		
Synopsis	Starts process of timer event synchronized SSI communication.	
Header	r_ssi_rzt2_if.h	
Declaration	int32_t R_SSI_Control(const int32_t id, const r_ssi_cmd_t cmd, const void *p_buf);	
Description	This function enables the timer event synchronized SSI communication in the designated channel. In response to the communication, callback function registered by the R_SSI_CMD_SET_CALLBACK command is called. See “4.5.1 ssi_int_ssi_callback” and “4.5.2 ssi_int_fifo_callback” for details of the callback functions. Command name is defined for compatibility with the RZ/T2M group SSI driver interface. The RZ/T2L group SSI driver does not support transmission trigger generation by using the event link controller (ELC). Alternatively, it generates transmission trigger without CPU intervention, by activating DMA with the timer event.	
Arguments	id	: Designates the ID to be used. (It is defined in r_ssi_rzt2_dat.h.)
		R_SSI0_ID : Specifies channel 0.
		R_SSI1_ID : Specifies channel 1.
		Others : Setting is not allowed.
	cmd	: Designates R_SSI_CMD_ELC_START.
	p_buf	: Not used. (Designate NULL.)
Return value	R_SSI_SUCCESS	: Normal termination
	R_SSI_ERR_ACCESS	: Abnormal termination (The channel is not open.)
	R_SSI_ERR_INVALID_ARG	: Abnormal termination (The id or cmd is invalid.)
	R_SSI_ERR_BUSY	: Abnormal termination (The data transfer or the timer event synchronized process is in progress.)
Note	Be sure to call R_SSI_Open before calling this function.	

(e) R_SSI_CMD_ELC_STOP

R_SSI_CMD_ELC_STOP		
Synopsis	Stops process of timer event synchronized SSI communication.	
Header	r_ssi_rzt2_if.h	
Declaration	int32_t R_SSI_Control(const int32_t id, const r_ssi_cmd_t cmd, const void *p_buf);	
Description	This function disables the timer event synchronized SSI communication in the designated channel.	
Arguments	id	: Designates the ID to be used. (It is defined in r_ssi_rzt2_dat.h.)
		R_SSI0_ID : Specifies channel 0.
		R_SSI1_ID : Specifies channel 1.
		Others : Setting is not allowed.
	cmd	: Designates R_SSI_CMD_ELC_STOP.
	p_buf	: Not used. (Designate NULL.)
Return value	R_SSI_SUCCESS	: Normal termination
	R_SSI_ERR_ACCESS	: Abnormal termination (The channel is not open.)
	R_SSI_ERR_INVALID_ARG	: Abnormal termination (The id or cmd is invalid.)
	R_SSI_ERR_BUSY	: Abnormal termination (The data transfer is in progress.)
Note	Be sure to call R_SSI_Open before calling this function.	

4.4.5 R_SSI_GetStatus

R_SSI_GetStatus			
Synopsis	Gets encoder status.		
Header	r_ssi_rzt2_if.h		
Declaration	int32_t R_SSI_GetStatus(const int32_t id, r_ssi_status_t *p_status);		
Description	This function gets SSI communication channel status This function stores SPSR register contents of the SPI to the designated address.		
Arguments	id	: Designates the ID to be used. (It is defined in r_ssi_rzt2_dat.h.)	
		R_SSI0_ID	: Specifies channel 0.
		R_SSI1_ID	: Specifies channel 1.
		Others	: Setting is not allowed.
	p_status	: Acquired status Designates pointer to the status structure. Result of the acquired status is stored in the structure r_ssi_status_t. See “4.10.1(3) r_ssi_status_t” for details.	
Return value	R_SSI_SUCCESS		: Normal termination
	R_SSI_ERR_INVALID_ARG		: Abnormal termination (The id is invalid, or p_status is NULL.)

4.4.6 R_SSI_GetRxd

R_SSI_GetRxd			
Synopsis	Gets result of reception data.		
Header	r_ssi_rzt2_if.h		
Declaration	int32_t R_SSI_GetRxd(const int32_t id, r_ssi_rxd_t *p_rxd);		
Description	Gets result of reception data from the designated SSI communication channel.		
Arguments	id	: Designates the ID to be used. (It is defined in r_ssi_rzt2_dat.h.)	
		R_SSI0_ID	: Specifies channel 0.
		R_SSI1_ID	: Specifies channel 1.
		Others	: Setting is not allowed.
	p_rxd	: Reception data Designates pointer to the reception data structure. Result of the acquired data is stored in the structure r_ssi_rxd_t. See “4.10.1(2) r_ssi_rxd_t” for details.	
Return value	R_SSI_SUCCESS		: Normal termination
	R_SSI_ERR_INVALID_ARG		: Abnormal termination (The id is invalid, or p_rxd is NULL.)

4.5 Specification of User-defined Functions

4.5.1 ssi_int_ssi_callback

ssi_int_ssi_callback	
Synopsis	Indicates reception error interrupt of SSI communication.
Header	-
Declaration	static void ssi_int_ssi_callback(r_ssi_status_t *p_status);
Description	This callback function is registered by the R_SSI_Control(R_SSI_CMD_SET_CALLBACK) function. This function is called when the reception error interrupt occurs in response to the SSI communication. This function is the context of the interrupt handler. To ensure interrupt responsiveness, return quickly. The function name is an example and can be selected freely.
Arguments	p_status : Acquired status Designates pointer to the status structure. Result of the acquired status is stored in the structure r_ssi_status_t.
Return value	None

4.5.2 ssi_int_fifo_callback

ssi_int_fifo_callback	
Synopsis	Indicates data reception interrupt of SSI communication.
Header	-
Declaration	static void ssi_int_fifo_callback(r_ssi_result_t *p_result);
Description	This callback function is registered by the R_SSI_Control(R_SSI_CMD_SET_CALLBACK) function. This function is called when the data reception is completed by the SSI communication. This function is the context of the interrupt handler. To ensure interrupt responsiveness, return quickly. The function name is an example and can be selected freely.
Arguments	p_result : Result of SSI data reception Designates pointer to the reception data storage area. Result of the acquired field data and status are stored in the structure r_ssi_result_t.
Return value	None

4.6 Interrupt Handlers

4.6.1 enc_ch0_rxi_isr

enc_ch0_rxi_isr	
Synopsis	Interrupt handler for completion of the channel 0 data reception
Header	-
Declaration	void enc_ch0_rxi_isr(void);
Description	This is the interrupt handler for the following factors of the SSI communication channel 0. 1. Completion of the FIFO data reception
Arguments	None
Return value	None

4.6.2 enc_ch0_eri_isr

enc_ch0_eri_isr	
Synopsis	Interrupt handler for reception error of the channel 0
Header	-
Declaration	void enc_ch0_eri_isr(void);
Description	This is the interrupt handler for the following factors of the SSI communication channel 0. 1. SSI status interrupt (mode fault error) 2. SSI status interrupt (overrun error) 3. SSI status interrupt (underrun error)
Arguments	None
Return value	None

4.6.3 enc_ch1_rxi_isr

enc_ch1_rxi_isr	
Synopsis	Interrupt handler for completion of the channel 1 data reception
Header	-
Declaration	void enc_ch1_rxi_isr(void);
Description	This is the interrupt handler for the following factors of the SSI communication channel 1. 1. Completion of the FIFO data reception
Argument	None
Return value	None

4.6.4 enc_ch1_eri_isr

enc_ch1_eri_isr	
Synopsis	Interrupt handler for reception error of the channel 1
Header	-
Declaration	void enc_ch1_eri_isr(void);
Description	This is the interrupt handler for the following factors of the SSI communication channel 1. 1. SSI status interrupt (mode fault error) 2. SSI status interrupt (overrun error) 3. SSI status interrupt (underrun error)
Argument	None
Return value	None

4.7 Interrupt

Interrupt requests used with the SSI driver are listed below.

Table 4.2 Interrupt for the SSI Driver

Interrupt	ID	Outline
SPI1_SPRI	329	It occurs when the data reception of channel 0 is completed.
SPI1_SPEI	332	It occurs when the SSI data reception error of channel 0 is detected.
SPI2_SPRI	334	It occurs when the data reception of channel 1 is completed.
SPI2_SPEI	337	It occurs when the SSI data reception error of channel 1 is detected.

4.8 Constants / Error Codes

Table 4.3 shows the list of tables for definitions of constants and error codes. See individual table for each definition.

Table 4.3 List of Tables for Definitions of Constants and Error Codes

Table Number	Contents
Table 4.4	Constants to be used in the SSI Driver (r_ssi_rzt2_dat.h)
Table 4.5	User-defined Constants to be used in the SSI Driver (r_ssi_rzt2_config.h)
Table 4.6	Bit Rate
Table 4.7	Error Code
Table 4.8	Specified Value by the SSI Communication Controller

Table 4.4 Constants to be used in the SSI Driver (r_ssi_rzt2_dat.h)

Constant Name	Setting	Description
R_SSI0_ID	0x01	Configuration ID for the SSI communication channel 0
R_SSI1_ID	0x02	Configuration ID for the SSI communication channel 1

Table 4.5 User-defined Constants to be used in the SSI Driver (r_ssi_rzt2_config.h)

Constant Name	Setting	Description
R_SSI_NFINTV_2MHz	0	Setting value to the NFINTV bit when the bit rate is 2 MHz.
R_SSI_NFINTV_1MHz	1	Setting value to the NFINTV bit when the bit rate is 1 MHz.
R_SSI_NFINTV_400kHz	2	Setting value to the NFINTV bit when the bit rate is 400 kHz.
R_SSI_NFINTV_300kHz	2	Setting value to the NFINTV bit when the bit rate is 300 kHz.
R_SSI_NFINTV_200kHz	3	Setting value to the NFINTV bit when the bit rate is 200 kHz.
R_SSI_NFINTV_100kHz	4	Setting value to the NFINTV bit when the bit rate is 100 kHz.
R_SSI_NFINTV_80kHz	4	Setting value to the NFINTV bit when the bit rate is 80 kHz.
R_SSI_NFSCNT_2MHz	9	Setting value to the NFSCNT bit when the bit rate is 2 MHz.
R_SSI_NFSCNT_1MHz	9	Setting value to the NFSCNT bit when the bit rate is 1 MHz.
R_SSI_NFSCNT_400kHz	11	Setting value to the NFSCNT bit when the bit rate is 400 kHz.
R_SSI_NFSCNT_300kHz	15	Setting value to the NFSCNT bit when the bit rate is 300 kHz.
R_SSI_NFSCNT_200kHz	11	Setting value to the NFSCNT bit when the bit rate is 200 kHz.
R_SSI_NFSCNT_100kHz	11	Setting value to the NFSCNT bit when the bit rate is 100 kHz.
R_SSI_NFSCNT_80kHz	15	Setting value to the NFSCNT bit when the bit rate is 80 kHz.

Table 4.6 Bit Rate

Constant Name	Setting	Description
R_SSI_2MHz	2000	Bit rate setting (2 MHz)
R_SSI_1MHz	1000	Bit rate setting (1 MHz)
R_SSI_400kHz	400	Bit rate setting (400 kHz)
R_SSI_300kHz	300	Bit rate setting (300 kHz)
R_SSI_200kHz	200	Bit rate setting (200 kHz)
R_SSI_100kHz	100	Bit rate setting (100 kHz)
R_SSI_80kHz	80	Bit rate setting (80 kHz)

Table 4.7 Error Code

Constant Name	Setting	Description
R_SSI_SUCCESS	0	Normal termination
R_SSI_ERR_INVALID_ARG	-1	Argument error
R_SSI_ERR_BUSY	-2	The API cannot be executed
R_SSI_ERR_ACCESS	-3	API execution order error

Table 4.8 Specified Value by the SSI Communication Controller

Constant Name	Setting	Description
R_SSI_FLD_MAX_NUM	8	Maximum number of fields
R_SSI_FLDSIZE_MAX_NUM	32	Maximum number of bits of each field

4.9 Fixed-width Integers

Table 4.9 lists the fixed-width integers for the sample code. These fixed-width integers are defined in the standard libraries.

Table 4.9 Fixed-width Integers for the Sample Program

Symbol	Description
int8_t	8-bit signed integer (defined in the standard libraries)
int16_t	16-bit signed integer (defined in the standard libraries)
int32_t	32-bit signed integer (defined in the standard libraries)
int64_t	64-bit signed integer (defined in the standard libraries)
uint8_t	8-bit unsigned integer (defined in the standard libraries)
uint16_t	16-bit unsigned integer (defined in the standard libraries)
uint32_t	32-bit unsigned integer (defined in the standard libraries)
uint64_t	64-bit unsigned integer (defined in the standard libraries)

4.10 Structures, Unions, and Enumerated Types

4.10.1 Structures

(1) r_ssi_param_t

Parameters for settings of the SSI communication

```
typedef struct
{
    uint16_t    bitrate;            Bit rate
                                    The value is reflected in the bit rate register (SPBR) of the SPI. See
                                    “Table 4.6 Bit Rate” for the value to be designated.
    uint8_t     fldnum;             Number of data fields (1 to 8)
                                    Designates the number of fields of reception data.
    uint8_t     fldsize[R_SSI_F     Field size (1 to 32)
                LD_MAX_NUM];        Designates number of bits in each field.
    uint8_t     gry[R_SSI_FLD       Gray code conversion (0 or 1)
                _MAX_NUM];          0: Disables Gray code / binary conversion at the field data acquisition
                                    1: Enables Gray code / binary conversion at the field data acquisition.
    uint8_t     rxd0sel;            Field number (0 to 7)
                                    Designates field number to be shown as the received data rxd0.
    uint8_t     rxd1sel;            Field number (0 to 7)
                                    Designates field number to be shown as the received data rxd1.
    uint8_t     nfintv;             Noise filter sampling intervals
                                    Note: This is reserved for driver interface compatibility with RZ/T2M
                                    group SSI driver. This setting value is not used for RZ/N2L.
    uint8_t     nfscnt              Noise filter sampling counts
                                    Note: This is reserved for driver interface compatibility with RZ/T2M
                                    group SSI driver. This setting value is not used for RZ/N2L.
} r_ssi_param_t
```

(2) r_ssi_rxd_t

SSI reception data

```
typedef struct
{
    uint32_t    rxd0;   The value of received data indicated by the parameter rxd0sel is stored here.
    uint32_t    rxd1    The value of received data indicated by the parameter rxd1sel is stored here.
} r_ssi_rxd_t
```

(3) r_ssi_status_t

Status of the SSI communication

See “SPSR: SPI status register” of the “RZ/N2L Group User’s Manual: Hardware” for the details of members.

```
typedef struct
{
    bool    fifoerr;    Status of mode fault error, overrun error, and underrun error (logical sum of
                        OVRF bit and MODF bit of the SPSR register) is stored.
                        (true: Error exists.)
    bool    errflg;     SSI data reception error status
                        (This bit is not used. This is always false.
                         Error flag at the tail end of SSI communication data cannot be stored.)
    bool    cont;       Shows that the communication is continued from previous one. (When
                        reception signal level just before the SSI data is low, the status is continued.)
                        (true: Continued.)
    bool    end;        Shows that the communication is completed. (SPRF bit of the SPSR register)
                        (true: Completed.)
    bool    standby     Shows that the communication is ready to start (stand-by state).
                        (Inverted IDLNF bit of the SPSR register)
                        (true; Stand-by.)
} r_ssi_status_t
```

(4) r_ssi_result_t

Result of the SSI communication

```
typedef struct
{
    r_ssi_status_t      status;             Status of the SSI communication
                                            See structure “r_ssi_status_t” for details.
    uint32_t            fdat[R_SSI_FLD      Received data
                        _MAX_NUM]           Received data of every field is stored.
} r_ssi_result_t
```

(5) r_ssi_callback_t

Callback functions for the SSI driver

```
typedef struct
{
    r_ssi_int_ssi_cb_t      cbadr_int_ssi;      Pointer to the callback function to be called when SSI
                                                reception error interrupt occur.
                                                See “4.5.1 ssi_int_ssi_callback” for details.
    r_ssi_int_fifo_cb_t     cbadr_int_fifo      Pointer to the callback function to be called when SSI data
                                                reception interrupt occur.
                                                See “4.5.2 ssi_int_fifo_callback” for details.
} r_ssi_callback_t
```

4.10.2 Unions

Not used.

4.10.3 Enumerated Types

(1) r_ssi_cmd_t

Enumeated types indicating control commands are shown in Table 4.10.

Table 4.10 r_ssi_cmd_t

Constant Name	Setting	Description
R_SSI_CMD_SET_PARAM	0	Sets parameter
R_SSI_CMD_SET_CALLBACK	1	Sets callback functions
R_SSI_CMD_START	2	Starts communication
R_SSI_CMD_ELC_START	3	Enables timer event synchronized communication
R_SSI_CMD_ELC_STOP	4	Disables timer event synchronized communication
R_SSI_CMD_MAX	5	Number of control commands

4.11 Description of the Sample Program

4.11.1 Operation Outline

This sample program handles the following processes.

1) Sets communication parameters by the console commands.
2) Perform communications by the console commands.
   a) Acquires data and status by the FIFO data reception interrupt.
   b) Stops data acquisition by the SSI reception error interrupt.
   c) Indicates received data and status to the console.
3) Perform communications periodically using GPT timer function by the console command.
   a) Acquires data and status multiple times by the FIFO data reception interrupt.
   b) Stops data acquisition by the SSI reception error interrupt.
   c) Indicates multiple received data and status to the console.
4) Perform communications periodically using DMA activated by timer event for triggering transmission.
   a) Acquires data and status multiple times by the FIFO data reception interrupt.
   b) Stops data acquisition by the SSI reception error interrupt.
   c) Indicates multiple received data and status to the console.

(1) System Block Diagram

Figure 4.1 shows a block diagram of the system.

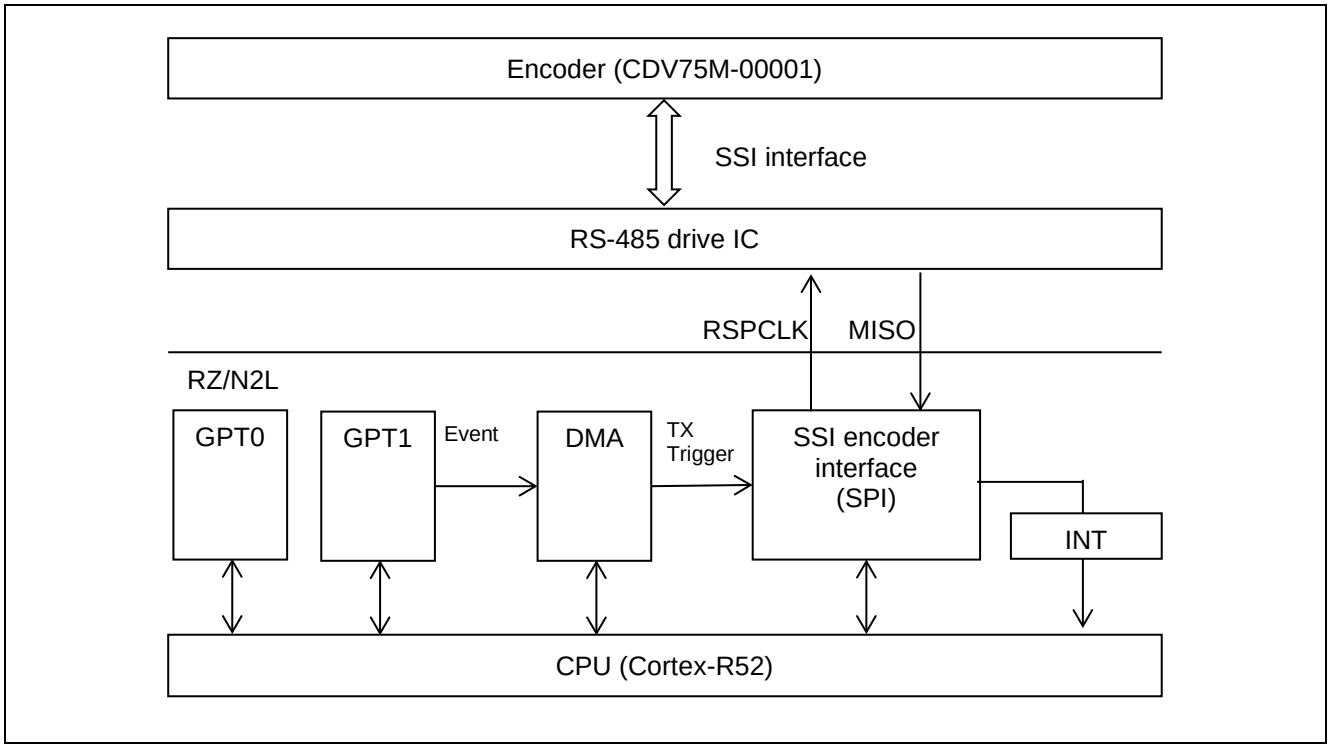


Figure 4.1 System Block Diagram

(2) Software Structure

Figure 4.2 shows the structure of the software.

The SSI driver has an opening process part composed of the R_SSI_Open function, a closing process part composed of the R_SSI_Close function, a sending requests part composed of the R_SSI_Control function, and a data reception part (interrupt handler) composed of callback functions.

The sample program has an SSI driver controller section that controls the SSI driver and sends requests, and a results indication section (callback) that displays the results of data reception.

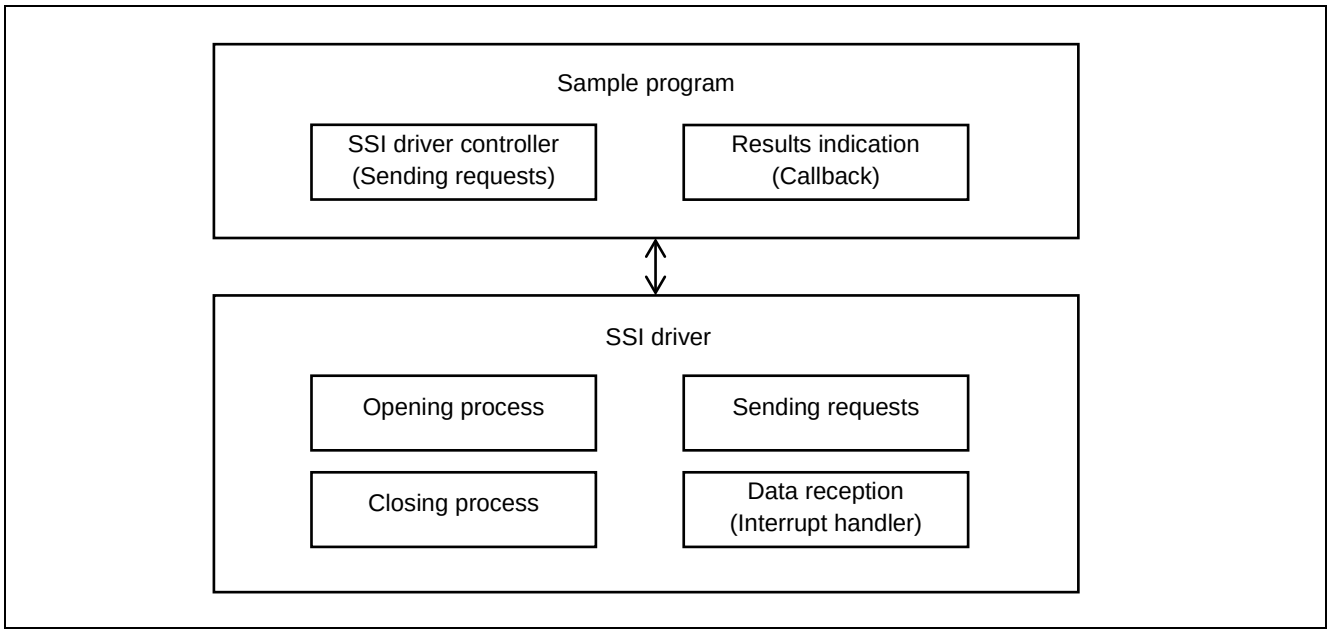


Figure 4.2 Software Structure

4.11.2 Functions of the Sample Program

Table 4.11 lists the major functions of the sample program.

Table 4.11 Major Functions of the Sample Program

Function Name	Page
hal_entry	23
enc_main	23
ssi_cmd_control	23
ssi_br	23
ssi_fld	24
ssi_gry	24
ssi_start	24
ssi_start_cont	24
ssi_timer_start	25
ssi_elc_start	25
ssi_exit	25
ssi_int_ssi_callback	25
ssi_int_fifo_callback	26
get_cmd	26
show_all	26
show_result	26
show_status	26
show_rxd	27
r_g_timer0_callback	27
timer_start	27
timer_stop	27

4.11.3 Specifications of the Sample Program Functions

(1) hal_entry

hal_entry	
Synopsis	Entry function of the SSI sample program
Header	–
Declaration	void hal_entry(void);
Description	This is the entry function of the SSI sample program. The function enc_main() is called from here.
Arguments	None
Return value	None

(2) enc_main

enc_main		
Synopsis	Main function of the SSI sample program	
Header	-	
Declaration	int32_t enc_main(uint8_t ch);	
Description	This is the main function of the SSI sample program. For details, see “4.11.6(1) Flowchart of enc_main”.	
Arguments	ch	Encoder channel number 0: Specifies channel 0., 1: Specifies channel 1.
Return value	0	: Normal termination
	Others	: Abnormal termination (error codes of the encoder interface)

(3) ssi_cmd_control

ssi_cmd_control	
Synopsis	Driver control function of the SSI sample program
Header	-
Declaration	static void ssi_cmd_control(void);
Description	This function performs the following processes. • Starting process of the encoder control • Console command input process • Ending process of the encoder control
Arguments	None
Return value	None

(4) ssi_br

ssi_br		
Synopsis	Function for console command “br”	
Header	-	
Declaration	static void ssi_br(uint32_t arg_num, char_t *p_arg[]);	
Description	This function is executed when the console command “br” is entered. See “4.11.8 Console Commands” for details.	
Arguments	arg_num	Number of strings entered from the console
	*p_arg[]	First address of string entered from the console
Return value	None	

(5) ssi_fld

ssi_fld	
Synopsis	Function for console command “fld”
Header	-
Declaration	static void ssi_fld(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command “fld” is entered. See “4.11.8 Console Commands” for details.
Arguments	arg_num Number of strings entered from the console *p_arg[] First address of string entered from the console
Return value	None

(6) ssi_gry

ssi_gry	
Synopsis	Function for console command “gry”
Header	-
Declaration	static void ssi_gry(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command “gry” is entered. See “4.11.8 Console Commands” for details.
Arguments	arg_num Number of strings entered from the console *p_arg[] First address of string entered from the console
Return value	None

(7) ssi_start

ssi_start	
Synopsis	Function for console command “start”
Header	-
Declaration	static void ssi_start(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command “start” is entered. See “4.11.8 Console Commands” for details.
Arguments	arg_num Number of strings entered from the console *p_arg[] First address of string entered from the console
Return value	None

(8) ssi_start_cont

ssi_start_cont	
Synopsis	Function for console command “start_cont”
Header	-
Declaration	static void ssi_start_cont(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command “start_cont” is entered. See “4.11.8 Console Commands” for details.
Arguments	arg_num Number of strings entered from the console *p_arg[] First address of string entered from the console
Return value	None

(9) ssi_timer_start

ssi_timer_start	
Synopsis	Function for console command “timer_start”
Header	-
Declaration	static void ssi_timer_start(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command “timer_start” is entered. See “4.11.8 Console Commands” for details.
Arguments	arg_num Number of strings entered from the console *p_arg[] First address of string entered from the console
Return value	None

(10) ssi_elc_start

ssi_elc_start	
Synopsis	Function for console command “elc_start”
Header	-
Declaration	static void ssi_elc_start(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command “elc_start” is entered. See “4.11.8 Console Commands” for details.
Arguments	arg_num Number of strings entered from the console *p_arg[] First address of string entered from the console
Return value	None

(11) ssi_exit

ssi_exit	
Synopsis	Function for console command “exit”
Header	-
Declaration	static void ssi_exit(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command “exit” is entered. See “4.11.8 Console Commands” for details.
Arguments	arg_num Number of strings entered from the console *p_arg[] First address of string entered from the console
Return value	None

(12) ssi_int_ssi_callback

ssi_int_ssi_callback	
Synopsis	Callback function for SSI driver reception error interrupt
Header	-
Declaration	static void ssi_int_ssi_callback(r_ssi_status_t *p_status);
Description	This is a callback function called by the SSI driver when SSI reception error occurs.
Arguments	*p_status First address of the RAM where the received status is stored
Return value	None

(13) ssi_int_fifo_callback

ssi_int_fifo_callback	
Synopsis	Callback function for SSI driver data reception interrupt
Header	-
Declaration	static void ssi_int_fifo_callback(r_ssi_result_t *p_result);
Description	This is a callback function called by the SSI driver when data reception interrupt occurs.
Arguments	*p_result First address of the RAM where the received data is stored
Return value	None

(14) get_cmd

get_cmd	
Synopsis	Function for acquiring the command
Header	-
Declaration	static uint32_t get_cmd(char_t *p_arg[], const uint32_t arg_max);
Description	This function acquires the input command.
Arguments	p_arg Address where the acquired command to be stored arg_max Maximum number of arguments
Return value	Number of command arguments

(15) show_all

show_all	
Synopsis	Function to show all received results
Header	–
Declaration	static void show_all(const uint32_t num);
Description	This function shows every field, status, and contents of the RXD registers of all received data.
Arguments	num Total number of received data to show
Return value	None

(16) show_result

show_result	
Synopsis	Function to show received results (fields and status)
Header	-
Declaration	static void show_result(const r_ssi_result_t *p_result, uint8_t fld_num);
Description	This function shows every field and status.
Arguments	*p_result Received results to show fld_num Number of fields to show
Return value	None

(17) show_status

show_status	
Synopsis	Function to show received result (status)
Header	-
Declaration	static void show_status(const r_ssi_status_t *p_status);
Description	This function shows contents of status.
Arguments	*p_status Status to show
Return value	None

(18) show_rxd

show_rxd	
Synopsis	Function to show received data (rxd0, rxd1)
Header	-
Declaration	static void show_rxd(const r_ssi_rxd_t *p_rxd);
Description	This function shows selected frame data as rxd0, rxd1 of the received result.
Arguments	*p_rxd rxd0, rxd1 data to show
Return value	None

(19) r_g_timer0_callback

r_g_timer0_callback	
Synopsis	Callback function for the timer operation of the sample program
Header	-
Declaration	void r_g_timer0_callback(timer_callback_args_t *p_args);
Description	This callback function is called at fixed intervals by the GPT0. SSI communication is requested periodically by the timer.
Arguments	*p_args Timer callback arguments of the r_gpt driver
Return value	None

(20) timer_start

timer_start	
Synopsis	Function to start periodical operation of timer in the sample program
Header	-
Declaration	static int32_t timer_start(timer_ctrl_t * const p_ctrl, timer_cfg_t const * const p_cfg, uint32_t usec);
Description	This function activates timer GPT (GPT0 or GPT1) designated by the arguments.
Arguments	*p_ctrl Control data of the r_gpt driver *p_cfg Configuration data of the r_gpt driver usec Timer interval [usec]
Return value	R_SSI_SUCCESS : Normal termination R_SSI_ERR_ACCESS : Abnormal termination

(21) timer_stop

timer_stop	
Synopsis	Function to stop periodical operation of timer in the sample program
Header	-
Declaration	static int32_t timer_stop(timer_ctrl_t * const p_ctrl);
Description	This function deactivates timer GPT (GPT0 or GPT1) designated by the argument.
Arguments	*p_ctrl Control data of the r_gpt driver
Return value	R_SSI_SUCCESS : Normal termination R_SSI_ERR_ACCESS : Abnormal termination

4.11.4 Variables of the Sample Program

Table 4.12 shows the static variables.

Table 4.12 Static Variables

Type	Variable Name	Description
enum ssi_mode	ssi_mode	Holds operation mode of the SSI communication. SSI_MODE_SINGLE: Single time operation mode SSI_MODE_CONT: 2 consecutive times operation mode SSI_MODE_TIMER: Multiple times operation mode
r_ssi_param_t	ssi_param	Holds setting parameters of the SSI driver.
r_ssi_callback_t	ssi_callback_dat	Holds callback settings to the SSI driver.
r_ssi_status_t	ssi_status	Holds SSI status of the received results.
r_ssi_result_t	ssi_result[SSI_TIMER_COUNT_MAX]	Holds received results. SSI_TIMER_COUNT_MAX times of results at most are stored.
r_ssi_rxd_t	ssi_rxd[SSI_TIMER_COUNT_MAX]	Holds received results. SSI_TIMER_COUNT_MAX times of results at most are stored.
uint16_t	ssi_result_index	Indicates index to store received result.
int32_t	ssi_err_code	Holds error code of the SSI communication.

4.11.5 Constants of the Sample Program

Table 4.13 shows major constants used in the sample program.

Table 4.13 Major Constants

Constant Name	Setting	Description
SSI_CONT_COUNT_MAX	2	Number of consecutive communication times by the start_cont command.
SSI_TIMER_COUNT_MAX	5	Number of multiple communication times by the timer_start command and the elc_start command.
SSI_TIMER_PERIOD	1000	Default communication intervals by the start_cont, timer_start or elc_start commands in [usec].

4.11.6 Flowchart of the Major Processes

(1) Flowchart of enc_main

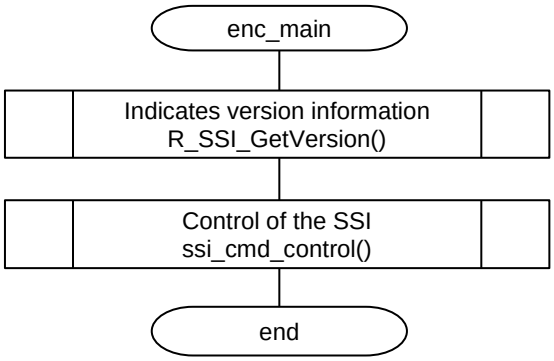


Figure 4.3 Flowchart of enc_main function

(2) Flowchart of ssi_cmd_control

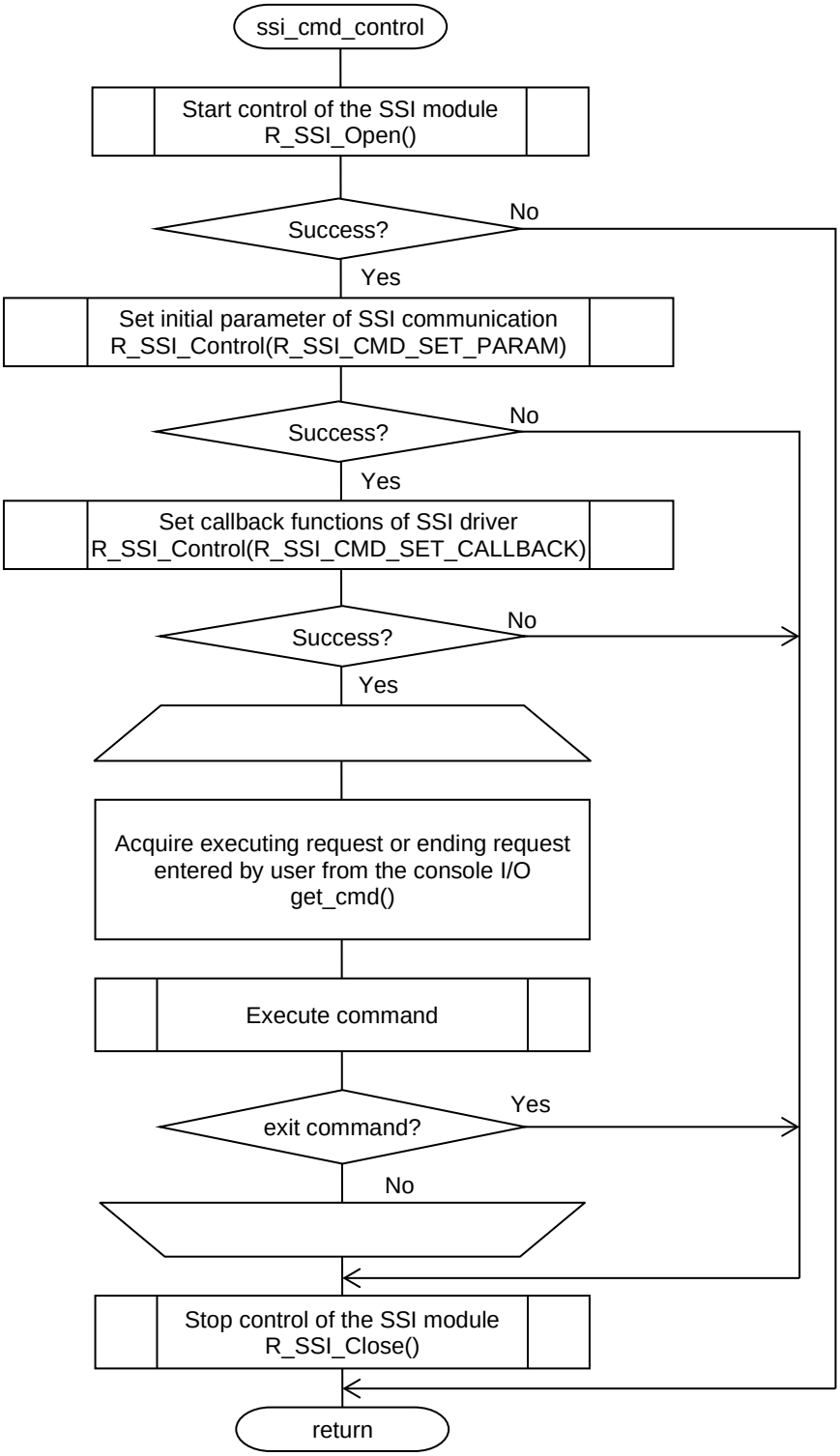


Figure 4.4 Flowchart of ssi_cmd_control function

(3) Flowchart of ssi_br, ssi_fld and ssi_gry

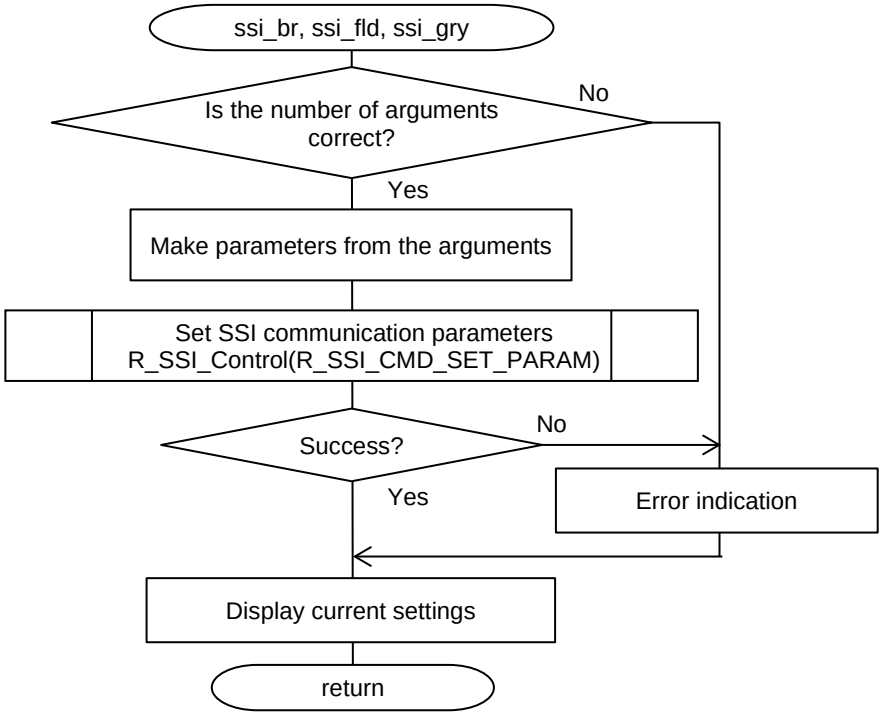


Figure 4.5 Flowchart of ssi_br, ssi_fld and ssi_gry functions

(4) Flowchart of ssi_start

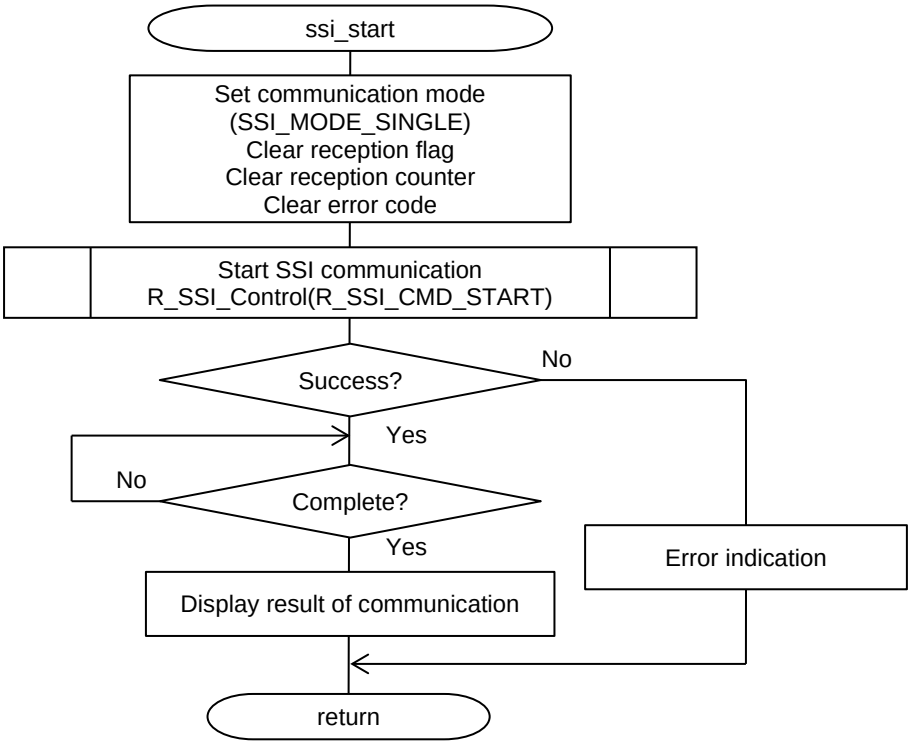


Figure 4.6 Flowchart of ssi_start function

(5) Flowchart of ssi_start_cont and ssi_timer_start

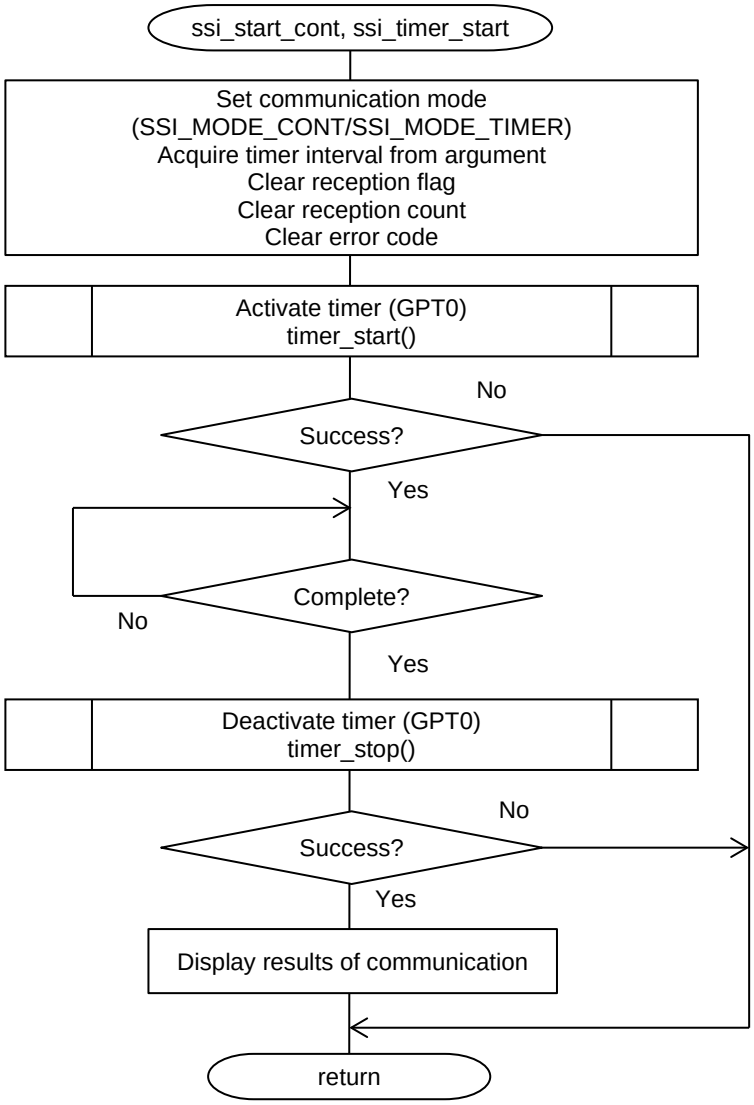


Figure 4.7 Flowchart of ssi_start_cont and ssi_timer_start functions

(6) Flowchart of ssi_elc_start

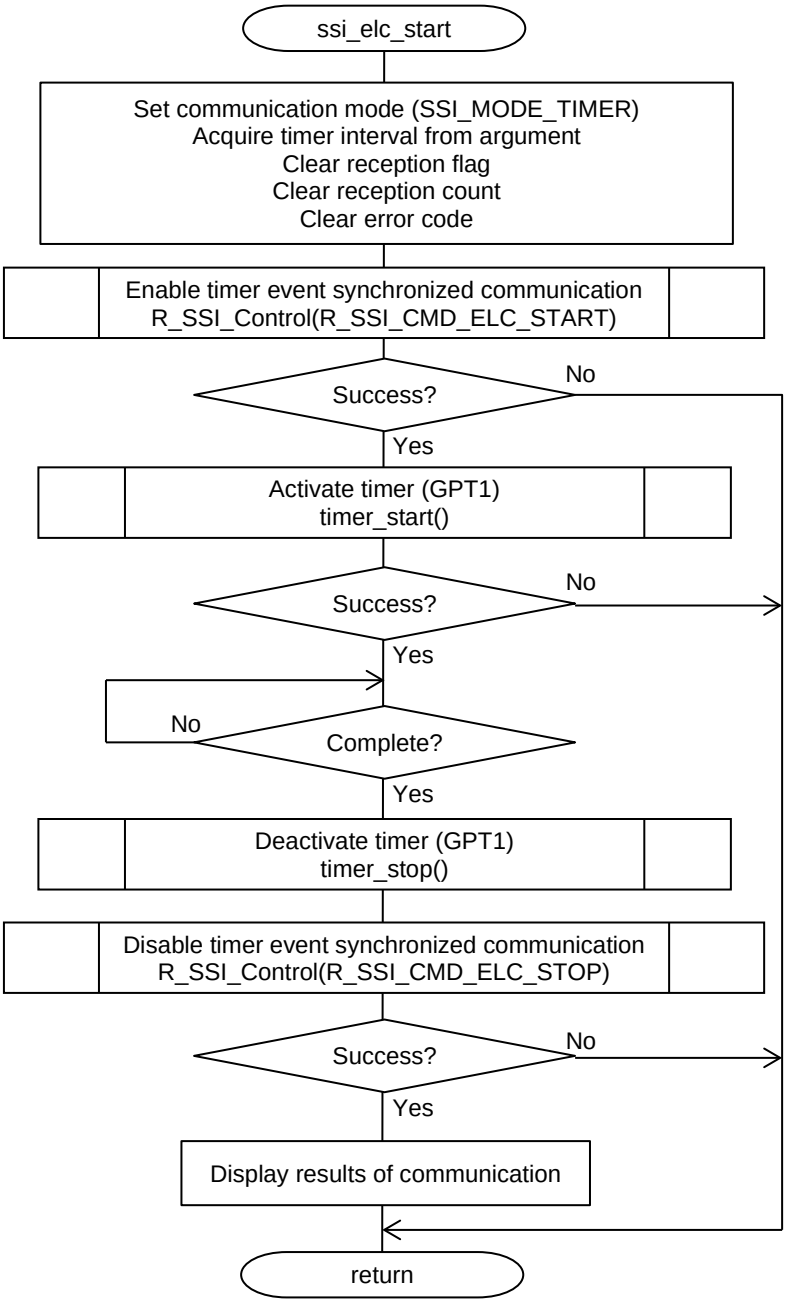


Figure 4.8 Flowchart of ssi_elc_start function

(7) Flowchart of ssi_exit

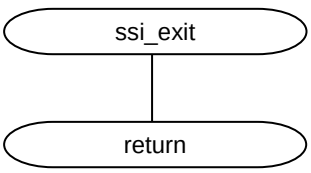


Figure 4.9 Flowchart of ssi_exit function

(8) Flowchart of ssi_int_ssi_callback

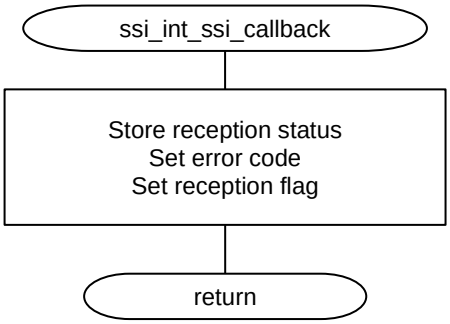


Figure 4.10 Flowchart of ssi_int_ssi_callback function

(9) Flowchart of ssi_int_fifo_callback

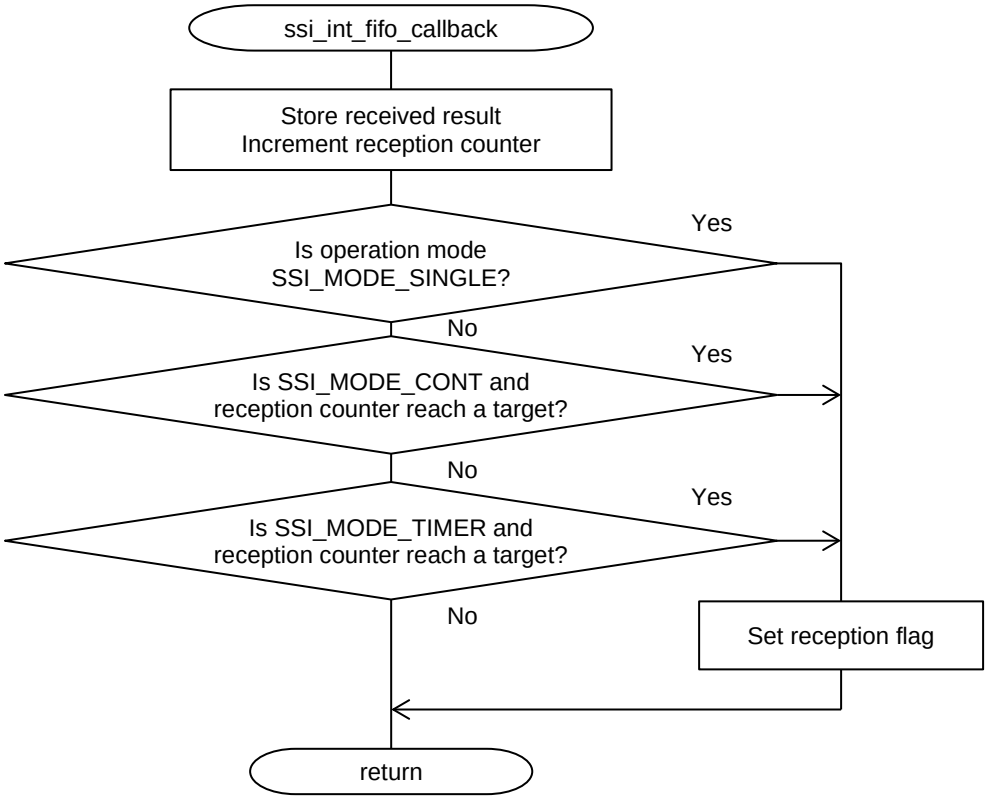


Figure 4.11 Flowchart of ssi_int_fifo_callback function

(10) Flowchart of get_cmd

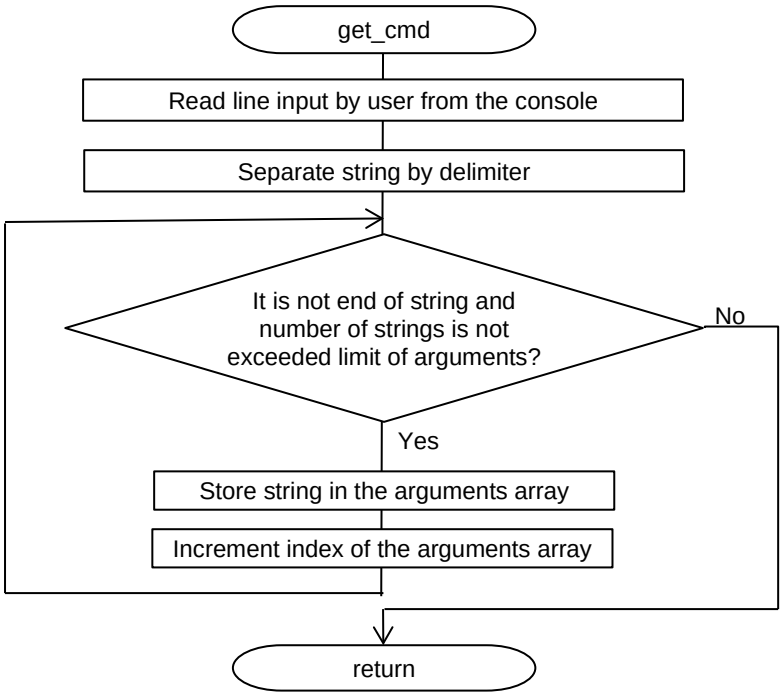


Figure 4.12 Flowchart of the get_cmd function

(11) Flowchart of show_all

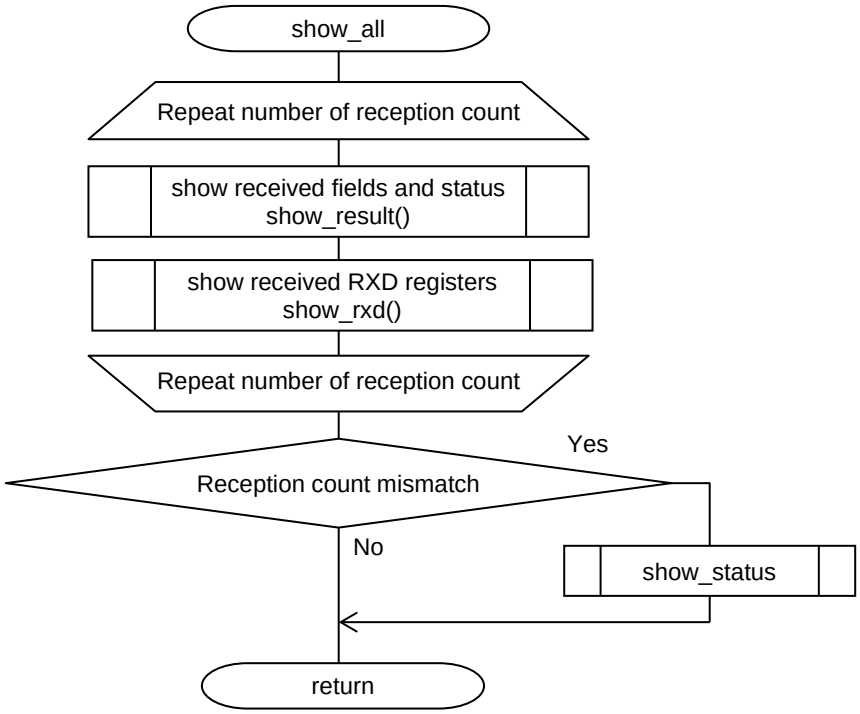


Figure 4.13 Flowchart of show_all function

(12) Flowchart of show_result, show_status and show_rxd

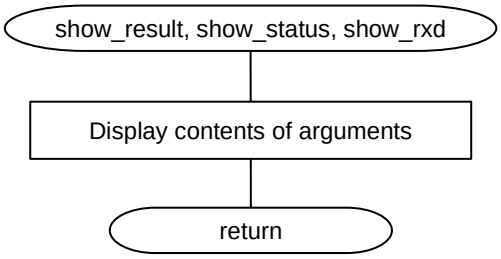


Figure 4.14 Flowchart of show_result, show_status and show_rxd functions

(13) Flowchart of r_g_timer0_callback

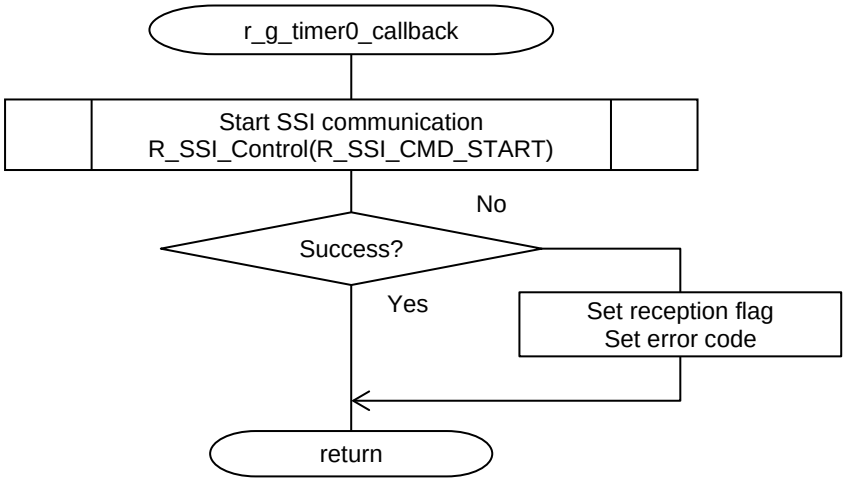


Figure 4.15 Flowchart of r_g_timer0_callback function

(14) Flowchart of timer_start

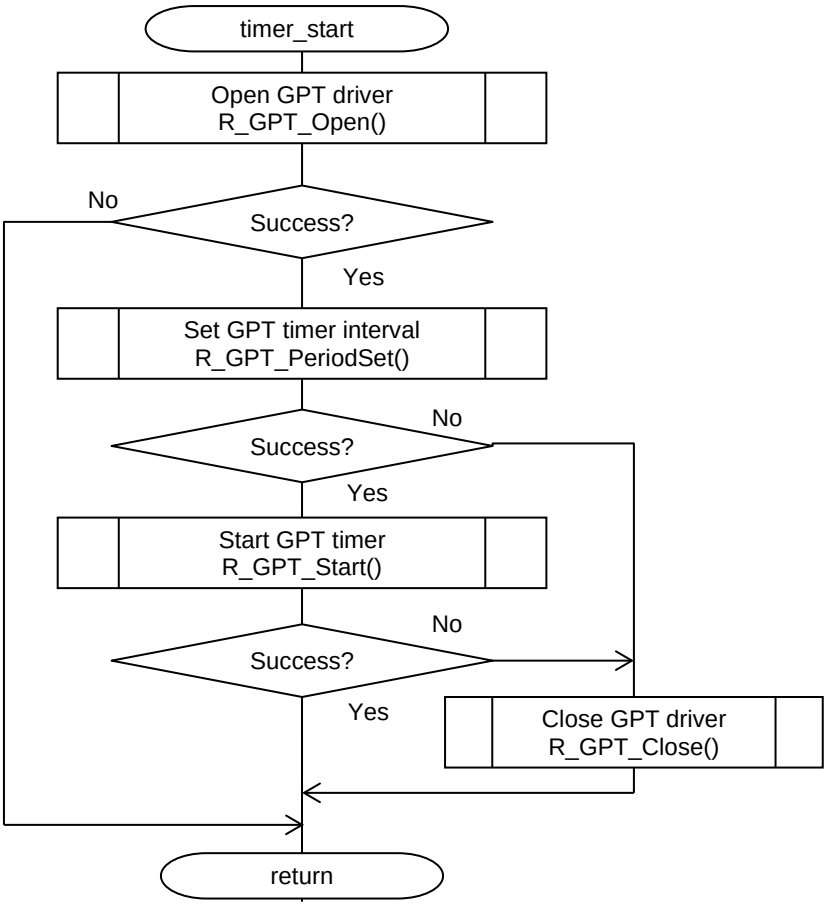


Figure 4.16 Flowchart of timer_start function

(15) Flowchart of timer_stop

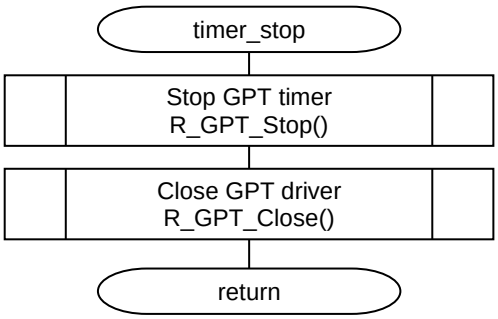


Figure 4.17 Flowchart of timer_stop function

4.11.7 Operation Sequence

(1) Startup Sequence

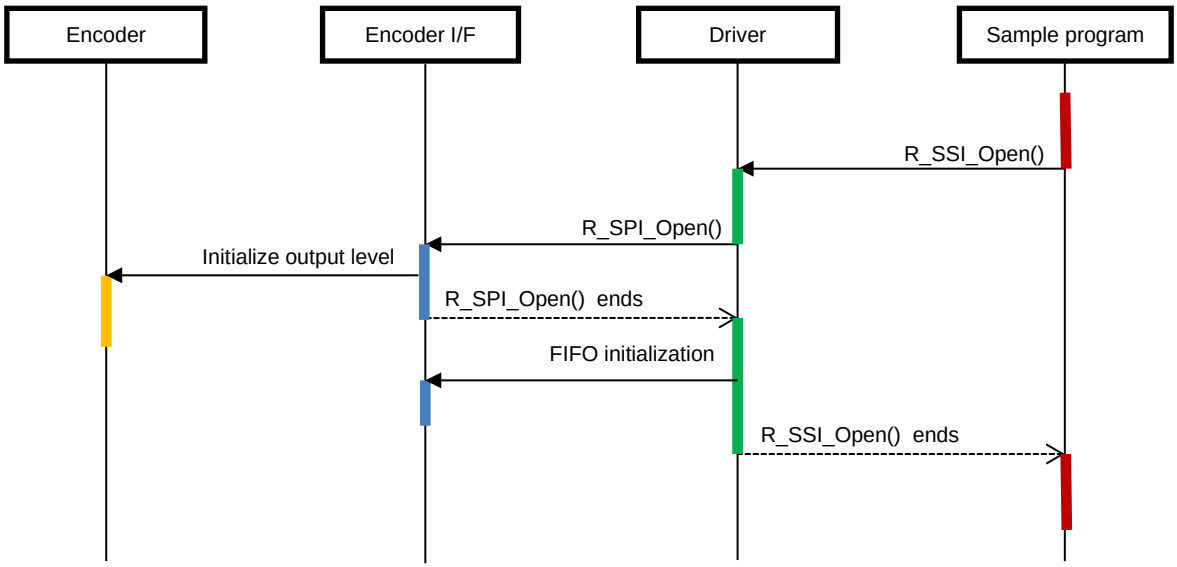


Figure 4.18 Startup Sequence Diagram

(2) Request Data Reception Sequence

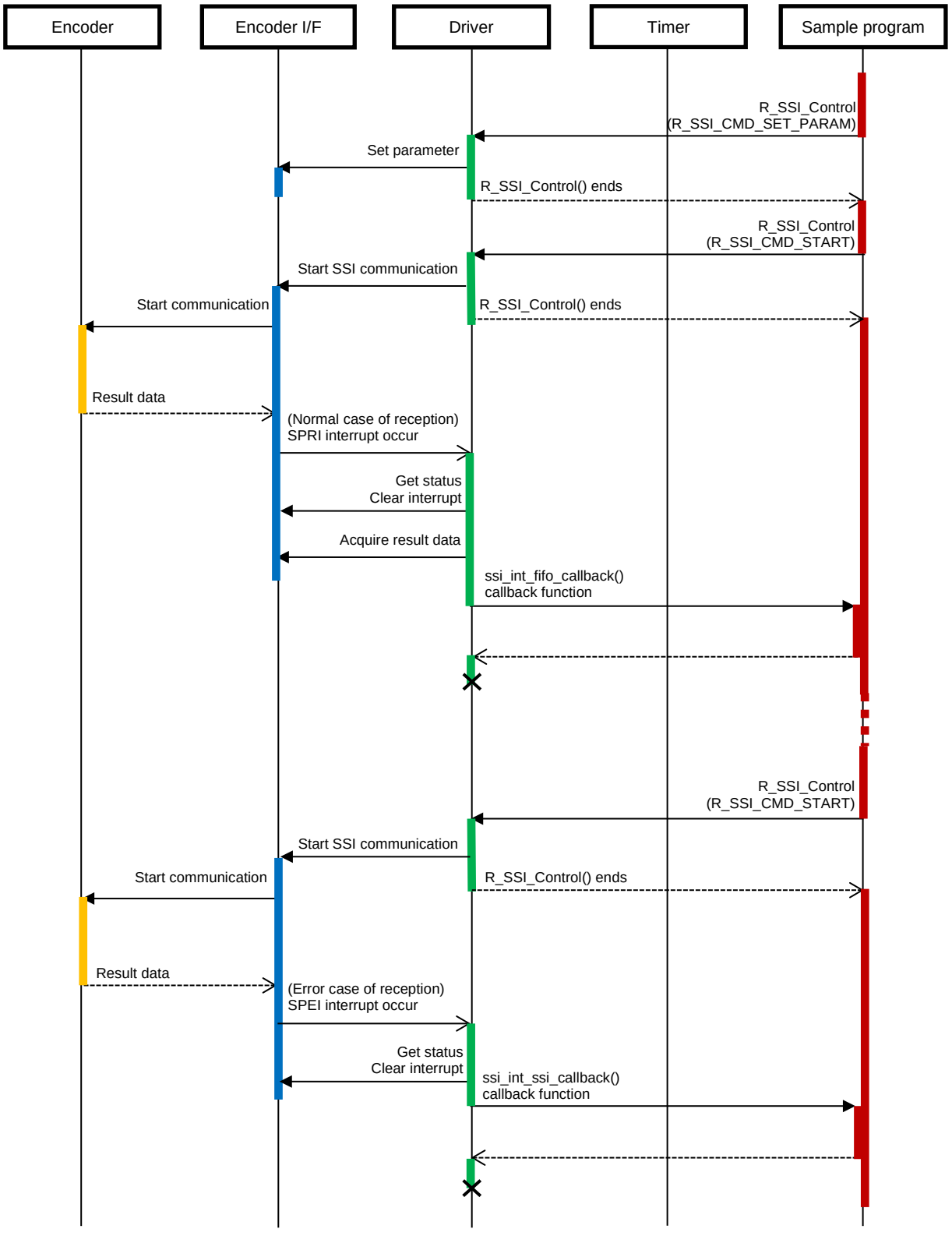
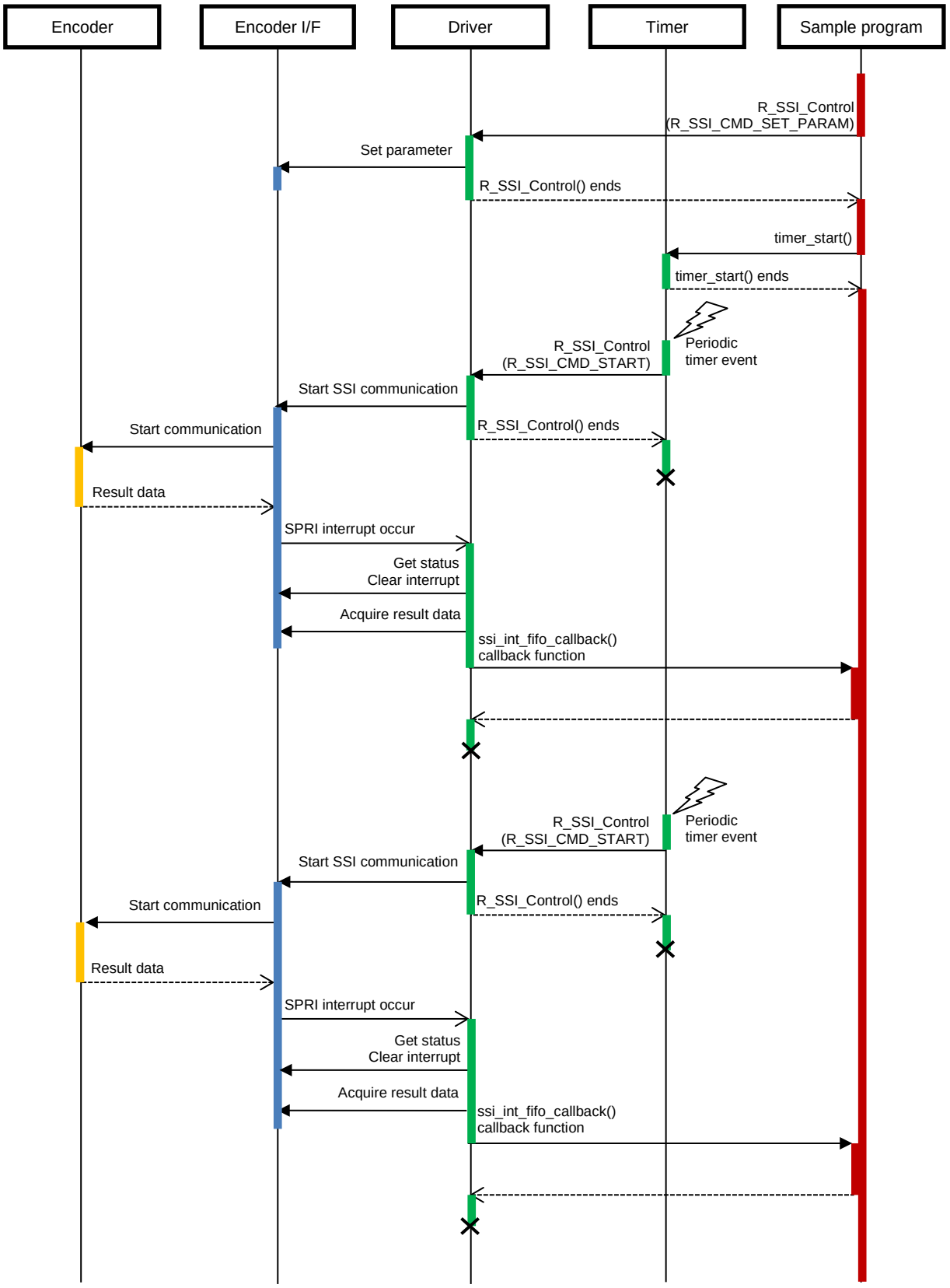


Figure 4.19 Request Data Reception Sequence Diagram

(3) Request Data Reception (Timer Operation) Sequence



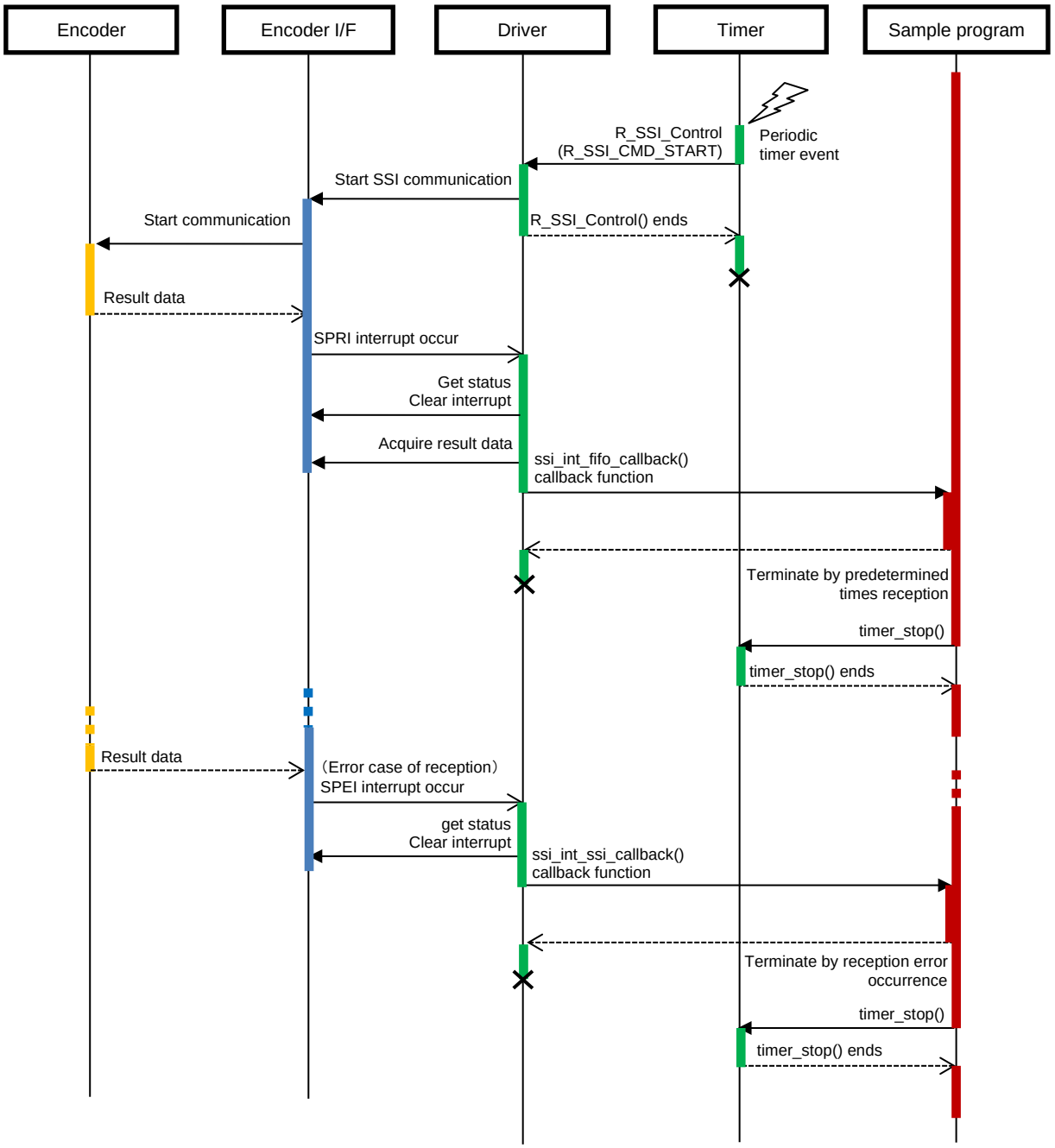
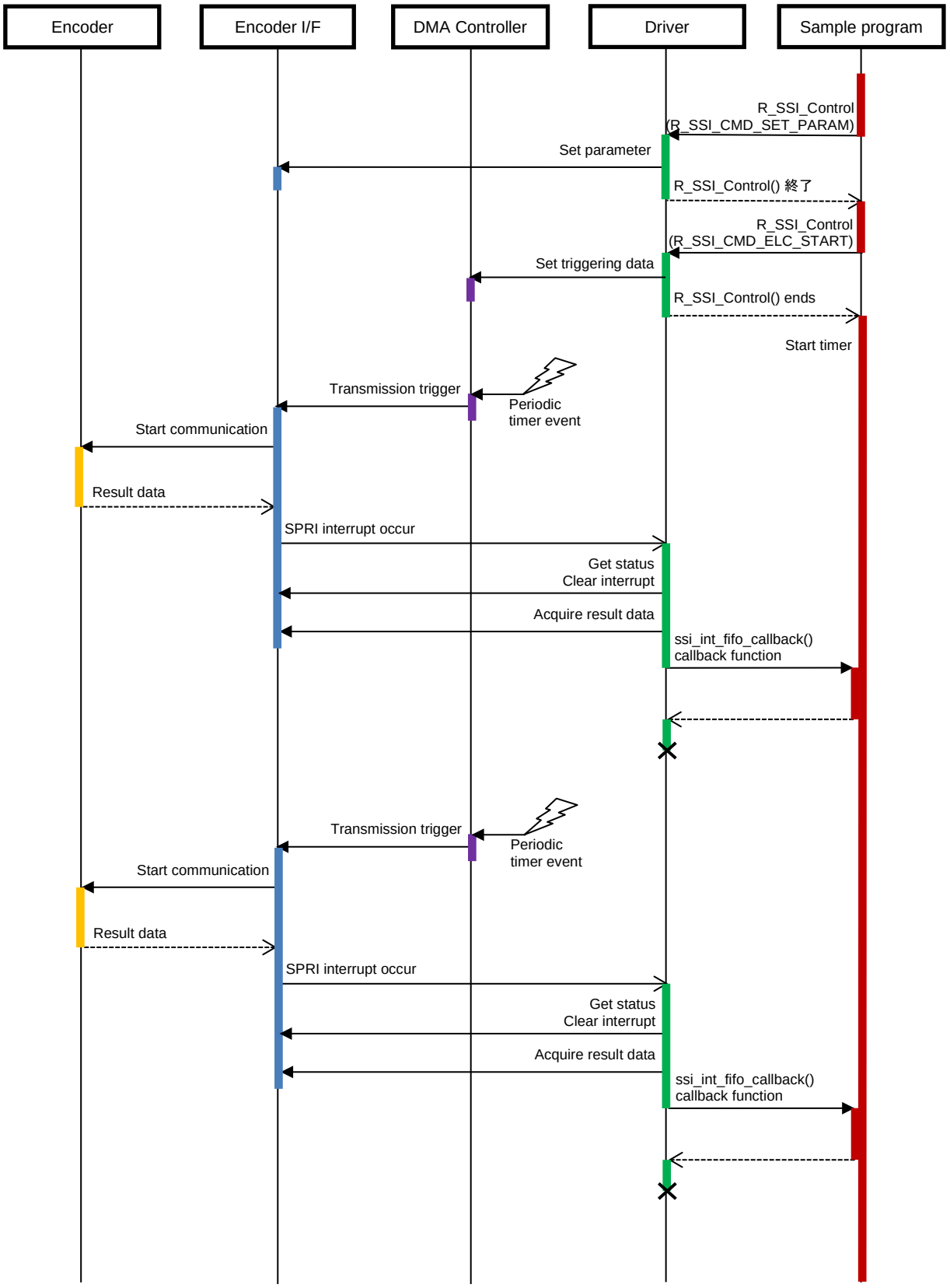


Figure 4.20 Request Data Reception (Timer Operation) Sequence Diagram

(4) Request Data Reception (Timer Event Synchronized Operation) Sequence



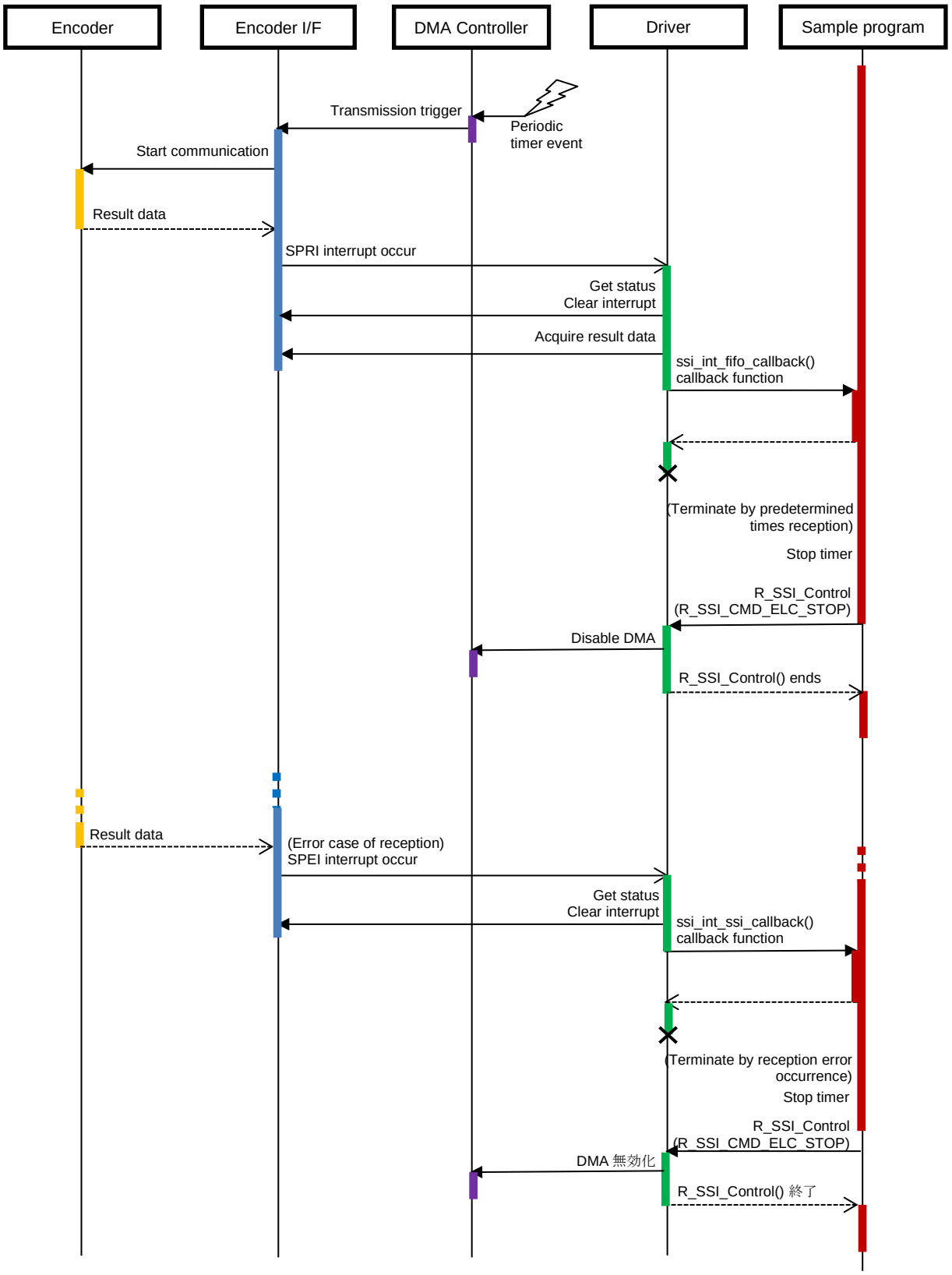


Figure 4.21 Request Data Reception (Timer Event Synchronized Operation) Sequence Diagram

(5) Stop Sequence

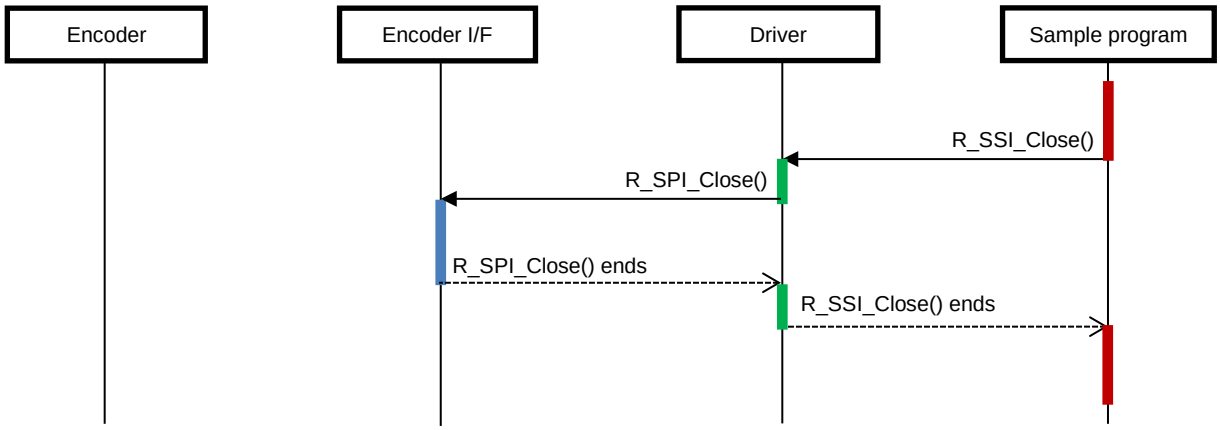


Figure 4.22 Stop Sequence Diagram

4.11.8 Console Commands

This sample program supports TR electronic encoder “CDV75M-00001” and Kübler encoder "8.5853FS3.1232.B723". The commands available for input from the console are listed below.

Table 4.14 Console Commands

Command	Description
br [bitrate]	Sets bitrate. bitrate: Designates bitrate. If no argument is designated, list of available arguments and current bitrate settings are displayed.
fld [fldsize0] [fldsize1] .. [fldsize7]	Sets number of fields and bit size of each field. Number of fields is specified by the number of designated field size. fldsize0: Designates the 0th field size. fldsize1: Designates the 1st field size. … fldsize7: Designates the 7th field size. If no argument is designated, current settings are displayed.
gry [gry0] [gry1] .. [gry7]	Sets Gray code / binary conversion information. gry0: Designates conversion of the 0th field. (1: enable conversion, 0: disable) gry1: Designates conversion of the 1st field. (1: enable conversion, 0: disable) … gry7: Designates conversion of the 7th field. (1: enable conversion, 0: disable) If no argument is designated, current settings are displayed.
start	Performs communication with the encoder and display result.
start_cont [period]	Performs two consecutive communications with the encoder and displays the results. period: Designates interval of the communications in [usec]. If no argument is designated, 1000 [usec] is used as the default interval.
timer_start [period]	Performs five consecutive communications with the encoder and displays the results. period: Designates interval of the communication in [usec]. If no argument is designated, 1000 [usec] is used as the default interval.
elc_start [period]	Performs five consecutive timer event synchronized communications with the encoder and displays the results. Event link controller (ELC) is not used. Alternatively, DMA activated by timer event is used to trigger transmission without CPU intervention. period: Designates interval of the communication in [usec]. If no argument is designated, 1000 [usec] is used as the default interval.
exit	Terminates program.

(1) Execution Result

After running, it will display the command prompt following the version. Enter the command after “ssi >”.

```
SSI sample program start
R_SSI_GetVersion = 4.0
ssi >
```

(2) Example of Command Execution

It is an example of executing console command “start”. Status and angular information are reported as the response from the encoder.

```
ssi >start
start command
- 0 -------------------------
FLD0 : 0x0000006D
FLD1 : 0x00000E8B
FLD2 : 0x00000000
FLD3 : 0x00000000
FLD4 : 0x00000001
FLD5 : 0x00000037
FIFOERR : 0
ERRFLG  : 0
CONT    : 0
END     : 1
STANDBY : 0
RXD0:FLD0 12bit : 0x0000006D
RXD1:FLD1 13bit : 0x00000E8B

ssi >
```

5. Sample Code

The sample code is available from the Renesas Electronics website.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar 31.23	-	First Edition issued.
2.00	Jul 07.24	4, 5, 6, 15, 9 to11, 19 15, 20, 25, 28, 27, 30, 33, 42, 43 45	Update Table 1.1 by supporting event synchronized operation. Correct notation of the board name. Update Table 3.1 and Table 4.2 by changing SPI channels for communication. Add commands for timer event synchronized communication and update related description of commands. Remove definitions for interrupt priority from Table 4.5. Update operation outline and block diagram. Add functions for timer event synchronized communications and update description of constants. Update description for timer_start and timer_stop functions. Correct ssi_cmd_control flowchart. Add ssi_elc_start flowchart. Add sequence diagram for timer event synchronized operation. Add elc_start console command.
3.00	Nov 7.25	1, 4, 5 8 to 14 45, 46	Change description for trademarks. Revise to unify description for functions. Add example of command execution.
4.00	Jun 26.26	9 to 13 23 to 27 44	Change prefix of pointer variables to “p_”. Update the SSI driver version.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit 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. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. 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 the 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.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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.

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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. 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 shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended 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; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall 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 shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

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