RZ/T2M Group

FA-CODER Sample Program

Summary

This application note explains a sample program for acquiring and indicating information from Tamagawa Seiki positional encoders by using the Encoder I/F Configuration Library (“EC-Lib”) of the RZ/T2M.

The major features of the program are listed below.
・For positional encoders from Tamagawa Seiki (FA-CODER®) with up to eight data fields.
・Readouts and indicates rotation angles, etc. from positional encoders.

Target Device

RZ/T2M

Table of 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
Encoder I/F (FA-CODER mode)	Transfer to and from FA-CODER
Interrupt controller (ICU)	Encoder I/F interrupt control
General PWM Timer (GPT) channel 0	Generation of event cycles to be input to ELC
Event Link Controller (ELC)	Link events output by GPT channel 0 to Encoder I/F
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.

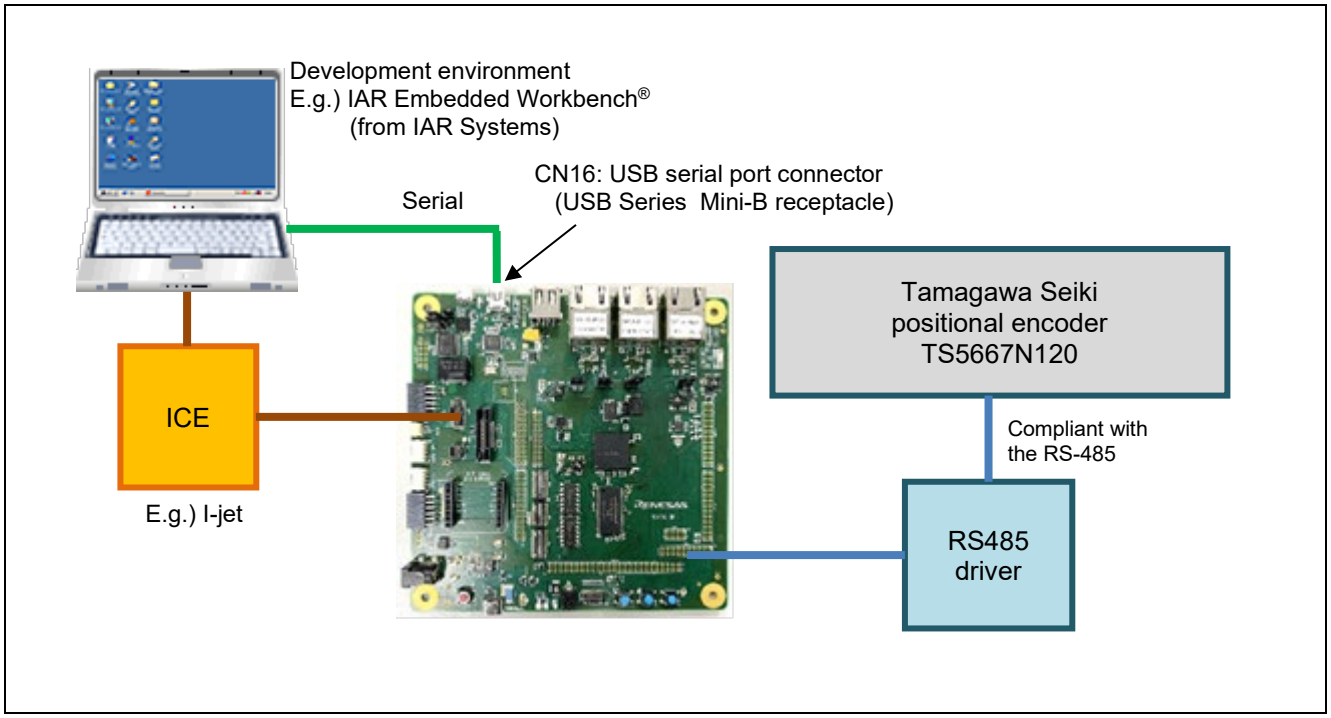


Figure 1.1 Operating Environment

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/T2M Group
Operating frequency	CPUCLK = 800MHz
Operating voltage	1.1V(Core) / 1.8V(PLL, etc.) / 3.3V(I/O)
Integrated development environment *	IAR Systems: Embedded Workbench® for Arm®
	RENESAS: e² studio
Board	RSK+RZT2M (RTK9RZT2M0C00000BE)
Devices (function to be used on the board)	None

Note: Refer to the RZ/T2M Group Encoder I/F FA-CODER sample program Release Note to check the version number of the integrated development environment.

3. Peripheral Modules

The basics of the peripheral modules, operating modes, and registers are described in the “RZ/T2M 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 (Function Pin Name)	I/O Port	Input/Output	Description
FACODER0	SDAT0 (ENCIF0)	P01_6	Input	Data reception pin
	SRQ0 (ENCIF2)	P02_0	Output	Request output pin
	DE0 (ENCIF3)	P02_2	Output	Drive/receive control pin
FACODER1	SDAT1 (ENCIF5)	P17_3	Input	Data reception pin
	SRQ1 (ENCIF7)	P17_5	Output	Request output pin
	DE1 (ENCIF8)	P03_0	Output	Drive/receive control pin

4. Software

4.1 FA-CODER Driver Function

The functions of the FA-CODER driver are listed below.

1) Initializing the encoder interface, interrupt controller, and pins
2) Transmitting requests to the FA-CODER and receiving results
3) Notification of errors in transfer from the FA-CODER

4.2 File Structures

For the file structure, refer to the release note for the RZ/T2M Group Encoder I/F FA-CODER 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
FA-CODER driver API functions	R_FAC_Open	7
	R_FAC_Close	7
	R_FAC_GetVersion	8
	R_FAC_Control	8
User-defined functions	callback_req_result	11
	callback_e2prom_result	11
	callback_elctimer_result	12
Interrupt-handlers	enc_ch0_int_isr	12
	enc_ch1_int_isr	12

4.4 Specifications of API Functions

4.4.1 R_FAC_Open

R_FAC_Open	
Synopsis	Starting control of the encoder
Header	r_fac_rzt2_if.h
Declaration	r_fac_err_t R_FAC_Open(const int32_t id, const r_fac_info_t *p_info);
Description	This function handles starting of FA-CODER specified by the arguments.
Arguments	id : Specifies the ID to be used. (It is defined in r_fac_rzt2_dat.h.) R_FAC0_ID : Specifies channel 0. R_FAC1_ID : Specifies channel 1. Others : Setting is not allowed. p_info : Sets information about the encoder. Designate the address of the structure r_fac_info_t where encoder information is stored.
Return value	R_FAC_SUCCESS : Normal termination R_FAC_ERR_ACCESS : Abnormal termination (The driver is already opened.) R_FAC_ERR_INVALID_ARG : Abnormal termination (The id or p_info member is not specified.)
Note	Before executing this function, be sure to use the EC-Lib to configure and start the encoder interface.

4.4.2 R_FAC_Close

R_FAC_Close	
Synopsis	Ending control of the encoder
Header	r_fac_rzt2_if.h
Declaration	r_fac_err_t R_FAC_Close(const int32_t id);
Description	This function handles ending of the encoder interface driver.
Arguments	id : Specifies the ID to be used. (It is defined in r_fac_rzt2_dat.h.) R_FAC0_ID : Specifies channel 0. R_FAC1_ID : Specifies channel 1. Others : Setting is not allowed.
Return value	R_FAC_SUCCESS : Normal termination R_FAC_ERR_INVALID_ARG : Abnormal termination (The id is not specified.) R_FAC_ERR_BUSY : Abnormal termination (Transfer is in progress.)
Note	The encoder driver cannot be closed while transferring data to or from the encoder is in progress. If this function is executed while the connection is closed, it returns R_FAC_SUCCESS without any processing. Stop the encoder interface using EC-Lib after executing this function.

4.4.3 R_FAC_GetVersion

R_FAC_GetVersion		
Synopsis	Acquiring the version number of the encoder	
Header	r_fac_rzt2_if.h	
Declaration	uint32_t R_FAC_GetVersion(void);	
Description	This function acquires the version number of the encoder interface driver.	
Arguments	None	
Return value	Version information	: The major and the minor parts of the version number are stored in the sixteen MSBs and the sixteen LSBs, respectively. Ex.) For ver.1.2, the value returned is 0x00010002

4.4.4 R_FAC_Control

R_FAC_Control		
Synopsis	Controlling the encoder	
Header	r_fac_rzt2_if.h	
Declaration	r_fac_err_t R_FAC_Control(const int32_t id, const r_fac_cmd_t cmd, void *const p_buf);	
Description	This function controls operation of the encoder interface driver. The operation changes by specifying the command as argument cmd. For details of the operation of each command, see section “4.4.5 List of Control Commands”.	
Arguments	id	: Specifies the ID to be used. (It is defined in r_fac_rzt2_dat.h.)
		R_FAC0_ID : Specifies channel 0.
		R_FAC1_ID : Specifies channel 1.
		Others : Setting is not allowed.
	cmd	: Command
		For details, see section “4.10.3(2), r_fac_cmd_t”.
	p_buf	: Argument for each cmd
Return value	R_FAC_SUCCESS	: Normal termination
	R_FAC_ERR_ACCESS	: Abnormal termination (The channel is not open.)
	R_FAC_ERR_BUSY	: Abnormal termination (Transfer is in progress.)
	R_FAC_ERR_INVALID_ARG	: Abnormal termination (The id or cmd is not specified, or the p_buf is NULL.)

4.4.5 List of Control Commands

(1) R_FAC_CMD_REQ

R_FAC_CMD_REQ		
Synopsis	Sending requests to the FA-CODER	
Header	r_fac_rzt2_if.h	
Declaration	r_fac_err_t R_FAC_Control(const int32_t id, const r_fac_cmd_t cmd, void *const p_buf);	
Description	This function sends requests to the FA-CODER.	
Arguments	id	: Specifies the ID to be used. (It is defined in r_fac_rzt2_dat.h.)
		R_FAC0_ID : Specifies channel 0.
		R_FAC1_ID : Specifies channel 1.
		Others : Setting is not allowed.
	cmd	: R_FAC_CMD_REQ
	p_buf	: Request information
		Specifies the pointer to the r_fac_req_t structure which holds the request information. For details, see section “4.10.1(2), r_fac_req_t”.
Return value	R_FAC_SUCCESS	: Normal termination
	R_FAC_ERR_ACCESS	: Abnormal termination (The channel is not open.)
	R_FAC_ERR_INVALID_ARG	: Abnormal termination (The id or cmd is not specified, the p_buf is NULL or the structure r_fac_req_t member is not specified.)
	R_FAC_ERR_BUSY	: Abnormal termination (Transfer is in progress.)

(2) R_FAC_CMD_E2PROM

R_FAC_CMD_E2PROM		
Synopsis	Sending requests to the FA-CODER	
Header	r_fac_rzt2_if.h	
Declaration	r_fac_err_t R_FAC_Control(const int32_t id, const r_fac_cmd_t cmd, void *const p_buf);	
Description	This function sends requests to the FA-CODER.	
Arguments	id	: Specifies the ID to be used. (It is defined in r_fac_rzt2_dat.h.)
		R_FAC0_ID : Specifies channel 0.
		R_FAC1_ID : Specifies channel 1.
		Others : Setting is not allowed.
	cmd	: R_FAC_CMD_E2PROM
	p_buf	: E2PROM information
		Specifies the pointer to the r_fac_e2prom_data_t structure which holds the E2PROM information. For details, see section “4.10.1(3), r_fac_e2prom_data_t”.
Return value	R_FAC_SUCCESS	: Normal termination
	R_FAC_ERR_ACCESS	: Abnormal termination (The channel is not open.)
	R_FAC_ERR_INVALID_ARG	: Abnormal termination (The id or cmd is not specified, the p_buf is NULL or the structure r_fac_e2prom_data_t member is not specified.)
	R_FAC_ERR_BUSY	: Abnormal termination (Transfer is in progress.)

(3) R_FAC_CMD_ELCTIMER

R_FAC_CMD_ELCTIMER			
Synopsis	Send ELC event synchronized requests to the FA-CODER		
Header	r_fac_rzt2_if.h		
Declaration	r_fac_err_t R_FAC_Control(const int32_t id, const r_fac_cmd_t cmd, void *const p_buf);		
Description	This function sends ELC event synchronized requests to the FA-CODER.		
Arguments	id	: Specifies the ID to be used. (It is defined in r_fac_rzt2_dat.h.)	
		R_FAC0_ID	: Specifies channel 0.
		R_FAC1_ID	: Specifies channel 1.
		Others	: Setting is not allowed.
	cmd	: R_FAC_CMD_ELCTIMER	
	p_buf	: Request information Specifies the pointer to the r_fac_req_t structure which holds the request information. For details, see section “4.10.1(2) r_fac_req_t”.	
Return value	R_FAC_SUCCESS		: Normal termination
	R_FAC_ERR_ACCESS		: Abnormal termination (The channel is not open.)
	R_FAC_ERR_INVALID_ARG		: Abnormal termination (The id or cmd is not specified, the p_buf is NULL or the structure r_fac_req_t member is not specified.)
	R_FAC_ERR_BUSY		: Abnormal termination (Transfer is in progress.)

(4) R_FAC_CMD_ELCSTOP

R_FAC_CMD_ELCSTOP			
Synopsis	Stop sending ELC event synchronized requests to the FA-CODER		
Header	r_fac_rzt2_if.h		
Declaration	r_fac_err_t R_FAC_Control(const int32_t id, const r_fac_cmd_t cmd, void *const p_buf);		
Description	This function stops sending ELC event synchronized requests to the FA-CODER.		
Arguments	id	: Specifies the ID to be used. (It is defined in r_fac_rzt2_dat.h.)	
		R_FAC0_ID	: Specifies channel 0.
		R_FAC1_ID	: Specifies channel 1.
		Others	: Setting is not allowed.
	cmd	: R_FAC_CMD_ELCSTOP	
	p_buf	: Not used. Specify NULL.	
Return value	R_FAC_SUCCESS		: Normal termination
	R_FAC_ERR_ACCESS		: Abnormal termination (The channel is not open.)
	R_FAC_ERR_INVALID_ARG		: Abnormal termination

4.5 Specifications of User-defined Functions

4.5.1 callback_req_result

callback_req_result	
Synopsis	Callback function for notifying the results of data reception in response to transmission of requests to the FA-CODER
Header	-
Declaration	void callback_req_result(r_fac_result_t * p_result, uint8_t * p_rxdf);
Description	A callback function registered with the R_FAC_Control (R_FAC_CMD_REQ) function. This function conveys the results of data reception in response to requests. This function is the context of the interrupt handler. To ensure interrupt responsiveness, return immediately. The function name is an example and can be set freely.
Arguments	p_result : Result of reception See section “4.10.1(4), r_fac_result_t”. This structure is valid until the next command execution. If you need to refer to the structure after the next command execution, copy it to an appropriate memory area beforehand. p_rxdf : Received data This is an array of eight elements. The number of valid elements depends on the type of encoder and the transmission ID code. This array is valid until the next command execution. If you need to refer to the array after the next command execution, copy it to an appropriate memory area beforehand.
Return value	None

4.5.2 callback_e2prom_result

callback_e2prom_result	
Synopsis	Callback function for notifying the results of data reception in response to transmission of requests to the FA-CODER
Header	-
Declaration	void callback_e2prom_result(r_fac_result_t * p_result, uint8_t adf, uint8_t edf);
Description	A callback function registered with the R_FAC_Control (R_FAC_CMD_E2PROM) function. This function conveys the results of data reception in response to requests. This function is the context of the interrupt handler. To ensure interrupt responsiveness, return immediately. The function name is an example and can be set freely.
Arguments	p_result : Result of reception See section “4.10.1(4), r_fac_result_t”. This structure is valid until the next command execution. If you need to refer to the structure after the next command execution, copy it to an appropriate memory area beforehand. adf : Data received by the E2PROM (ADF) edf : Data received by the E2PROM (EDF)
Return value	None

4.5.3 callback_elctimer_result

callback_elctimer_result	
Synopsis	Callback function for notifying the results of data reception in response to transmission of requests to the FA-CODER
Header	-
Declaration	void callback_elctimer_result(r_fac_result_t * p_result, uint8_t *p_rxdf);
Description	Callback function registered with the R_FAC_Control (R_FAC_CMD_ELCTIMER) function. This function conveys the results of data reception in response to requests. This function is the context of the interrupt handler. To ensure interrupt responsiveness, return immediately. The function name is an example and can be set freely.
Arguments	p_result : Result of reception See section “4.10.1(4), r_fac_result_t”. This structure is valid until the next command execution. If you need to refer to the structure after the next command execution, copy it to an appropriate memory area beforehand. p_rxdf : Received data It is an array with 8 elements. The number of valid data depends on the encoder type and the transmission ID code. This array is valid until the next command execution. If you need to refer to the array after the next command execution, copy it to an appropriate memory area beforehand.
Return value	None

4.6 Interrupt Handler

4.6.1 enc_ch0_int_isr

enc_ch0_int_isr	
Synopsis	Interrupt handler for the completion of data reception from channel 0
Header	-
Declaration	void enc_ch0_int_isr(void);
Description	An interrupt handler for the data reception completed interrupt from the FA-CODER
Arguments	None
Return value	None

4.6.2 enc_ch1_int_isr

enc_ch1_int_isr	
Synopsis	Interrupt handler for the completion of data reception from channel 1
Header	-
Declaration	void enc_ch1_int_isr(void);
Description	An interrupt handler for the data reception completed interrupt from the FA-CODER
Arguments	None
Return value	None

4.7 Interrupts

Table 4.2 lists the interrupt for the FA-CODER driver.

Table 4.2 Interrupt for the FA-CODER Driver

Interrupt	ID	Outline
Channel 0 data reception	372	Completion of requested data reception (including error cases)
Channel 1 data reception	376	Completion of requested data reception (including error cases)

4.8 Constants and Error Codes

Table 4.3 lists the main constants for the FA-CODER driver (r_fac_rzt2.c). Table 4.4 lists the bit rate indices (r_fac_rzt2_if.h). The error codes are given in section “4.10.3(1) r_fac_err_t”.

Table 4.3 Main Constants for the FA-CODER Driver (r_fac_rzt2.c)

Constant Name	Setting Value	Description
FAC_ID_NUM	2	Total number of FA-CODER configuration IDs
FAC_CMD_NUM	4	Total number of FA-CODER driver commands
FAC_RXDF_MAX	8	Maximum data field number (Cannot be changed)

Table 4.4 Bit Rate Indices (r_fac_rzt2_if.h)

Constant Name	Setting Value	Description
R_FAC_2500KBPS	0	2.5 Mbps
R_FAC_5MBPS	1	5 Mbps
R_FAC_BITRATE_NUM	2	Total number of bit rate definitions

4.9 Fixed-width Integer Types

Table 4.5 lists the fixed-width integers for the sample code. These fixed-width integers are defined in the standard libraries.

Table 4.5 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

The main structures, union, and enumerated types are listed below.

4.10.1 Structures

(1) r_fac_info_t

Initialization information of the FA-CODER control unit

```
typedef struct
{
    uint8_t             bitrate;        Bit rate
                                        Designate the bit rate for communications with the encoder. See
                                        “Table 4.4 Bit Rate Indices (r_fac_rzt2_if.h)” for the values to be
                                        designated.
    uint16_t            rx_guard;       Reception guard band
                                        Designate period of the reception disabled band. The period is
                                        rx_guard×50 (ns). Use even value for rx_guard.
} r_fac_info_t
```

(2) r_fac_req_t

Request information for transmission to the FA-CODER

```
typedef struct
{
    uint8_t             txid;           Transmission ID codes (0 to 15)
    uint8_t             dfnum;          Number of data fields (1 to 8)
                                        Sets the number of data fields, which is uniquely determined by
                                        the type of encoder and transmission ID code.
    uint16_t            timotn;         Timeout period from the start of transmission to the completion of
                                        reception. The timeout period is timotn×50 (ns). The
                                        recommended value is 4000 (200 µs).
    r_fac_result_cb_t   p_result_cb;    Pointer to the callback function notifying the request results.
                                        For details, see section “4.5.1, callback_req_result”.
                                        If NULL is specified, the callback will not occur.
} r_fac_req_t
```

(3) r_fac_e2prom_data_t

Request information for transmission to the E2PROM

```
typedef struct
{
    uint16_t                timotn;         Timeout period from the start of transmission to the
                                            completion of reception
                                            The timeout period is timotn×50 (ns). The recommended
                                            value is 4000 (200 µs).
    uint8_t                 adr;            E2PROM address
    uint8_t                 data;           E2PROM data
    r_fac_e2prom_dir_t      dir;            Direction of the E2PROM read / write
                                            See section “4.10.3(4) r_fac_e2prom_dir_t“.
    r_fac_e2prom_result_cb_t p_result_cb;   Pointer to the callback function notifying the request
                                            results.
                                            For details, see section “4.5.2, callback_e2prom_result”.
                                            If NULL is specified, the callback will not occur.
} r_fac_e2prom_data_t
```

(4) r_fac_result_t

Status of the result of transfer from the FA-CODER

```
typedef struct
{
    r_fac_rx_err_t      result;     Result of reception
                                    For details, see section “4.10.3(3), r_fac_rx_err_t”.
    bool                rse;        Read sequence error *
    bool                ide;        Received ID error *
    bool                ebusy;      E2PROM access reception busy status bit
    uint8_t             rxid;       Received request ID
    uint8_t             rxidp;      Parity bit for the received request ID
    uint8_t             rxsfic;     Received information code
    uint8_t             rxsfea;     Received encoder alarm
    uint8_t             rxsfca;     Received communications alarm
    uint8_t             crc;        Received CRC data
    bool                conte;      Control field error *
    bool                crce;       CRC error *
    bool                sfome       Short form error *
    bool                timote;     Timeout error *
    bool                rxedfe;     Received EDF error *
    bool                rxadfe;     Received ADF error *
    bool                dfovfe;     Overflow error in the number of received data fields *
} r_fac_result_t
```

Note: The value is true when an error occurs.

4.10.2 Unions

Not used.

4.10.3 Enumerated Types

(1) r_fac_err_t

Error codes of the encoder interface

```
typedef enum
{
    R_FAC_SUCCESS            =0,    Normal termination
    R_FAC_ERR_INVALID_ARG    ,      Argument error
    R_FAC_ERR_BUSY           ,      State where API is not executable
    R_FAC_ERR_ACCESS                Error in the execution order of APIs
} r_fac_err_t
```

(2) r_fac_cmd_t

Command settings when the R_FAC_Control function is used

```
typedef enum
{
    R_FAC_CMD_REQ            ,      Sending requests to the encoder
    R_FAC_CMD_E2PROM         ,      Access to E2PROM
    R_FAC_CMD_ELCTIMER       ,      Start sending/receiving requests triggered by ELC event
                                    input
    R_FAC_CMD_ELCSTOP               Stop sending/receiving requests triggered by ELC event
                                    input
} r_fac_cmd_t
```

(3) r_fac_rx_err_t

Result of reception from the encoder

```
typedef enum
{
    R_FAC_SUCCESS            =0,    Normal termination
    R_FAC_ERR                       Error termination
} r_fac_rx_err_t
```

(4) r_fac_e2prom_dir_t

Direction of the E2PROM read / write

```
typedef enum
{
    R_FAC_E2PROM_READ        ,      For reading
    R_FAC_E2PROM_WRITE       ,      For writing
} r_fac_e2prom_dir_t
```

4.11 Description of the Sample Program

4.11.1 Operation Outline

This sample program handles the following processing.

1) Sending requests input from the console to the FA-CODER.
2) Indicating the data received from the FA-CODER in the console.
3) Send and receive commands using the ELC event trigger function of the encoder interface. (GPT events are linked as input events.)

(1) System Block Diagram

Figure 4.1 shows a block diagram of the system.

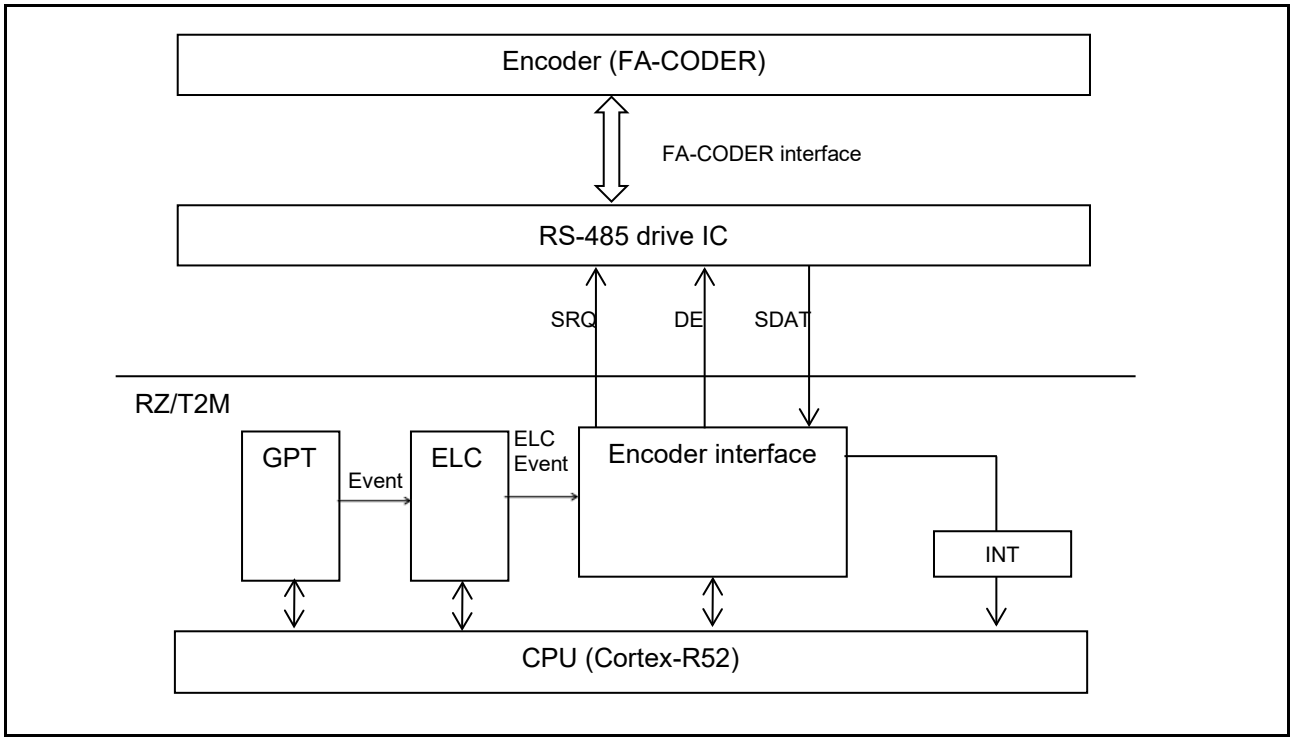


Figure 4.1 System Block Diagram

(2) Software Structure

Figure 4.2 is a block diagram of the software.

The FA-CODER driver has four sections: the opening process part configured of the function R_FAC_Open, the closing process part configured of the function R_FAC_Close, the request transmission part configured of the function R_FAC_Control, and the data reception part (interrupt handler) configured of the callback function.

The sample program has the FA-CODER driver controller section which controls the FA-CODER driver and sends requests, and the results indication section (callback) which indicates the result of data reception.

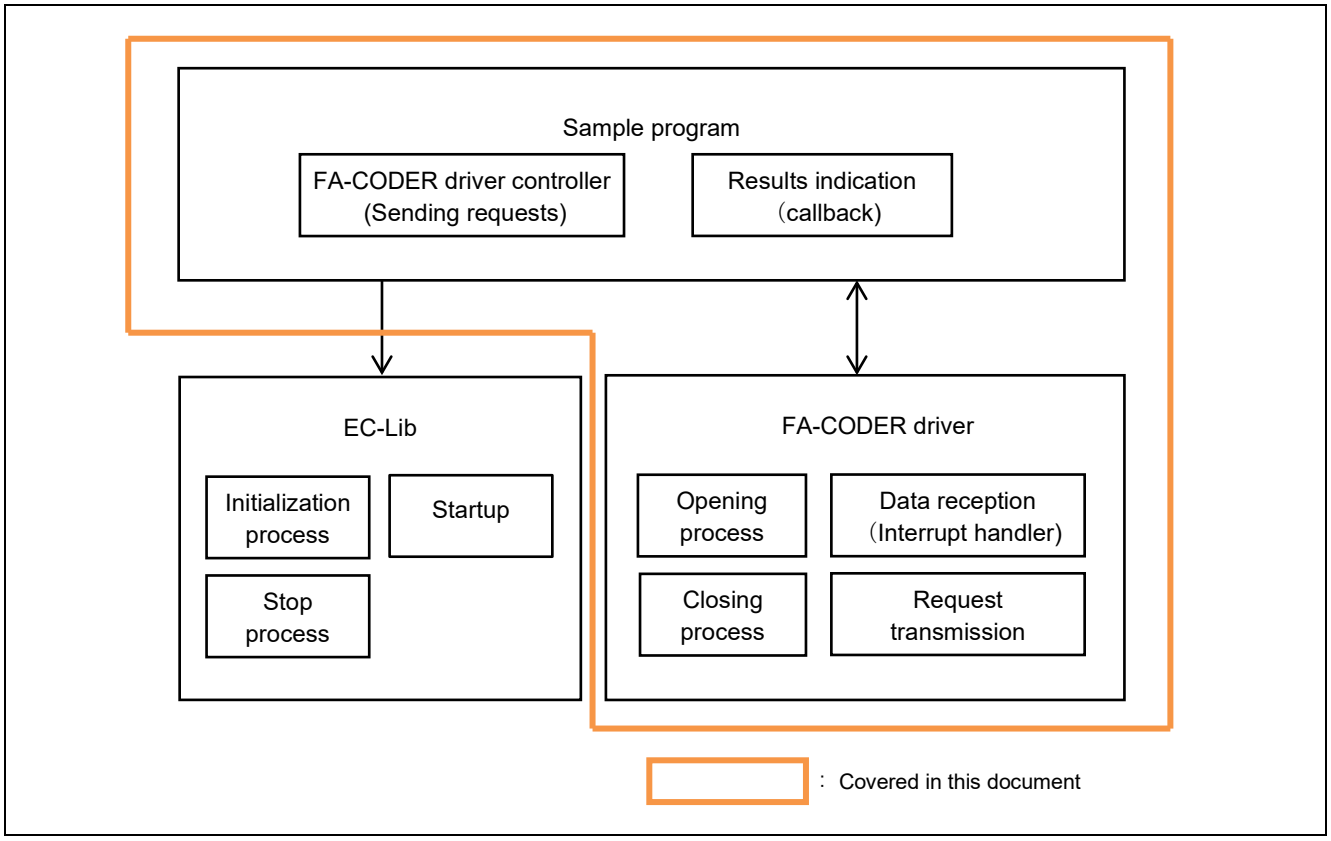


Figure 4.2 Software Structure

4.11.2 Sample Program Functions

Table 4.6 lists the main functions of the sample program.

Table 4.6 Main Functions of the Sample Program

Function Name	Page Number	
	Specification	Flowchart
hal_entry	20	–
enc_main	20	27
fac_get_cmd	20	28
fac_cmd_single	20	29
fac_cmd_multi	21	
fac_cmd_encid	21	
fac_cmd_req	21	30
fac_cmd_e2prom_write	21	31
fac_cmd_e2prom_read	22	32
fac_cmd_exit	23	36
fac_cmd_reset_single	22	33
fac_cmd_reset_multi	22	
fac_cmd_reset_all	22	
fac_cmd_elctimer	23	34
fac_cmd_elcstop	23	35
fac_trans_req	23	36
fac_trans_e2prom	24	
fac_trans_timer	24	37
fac_elc_stop	24	37
callback_req_result	24	38
callback_e2prom_result	25	
callback_elctimer_result	25	38

4.11.3 Specifications of Sample Program Functions

(1) hal_entry

hal_entry	
Synopsis	Entry function of the FA-CODER sample program
Header	-
Declaration	void hal_entry(void);
Description	This is the entry function of the FA-CODER 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 FA-CODER sample program	
Header	-	
Declaration	int32_t enc_main(uint8_t ch);	
Description	This is the main function of the FA-CODER sample program. For details, see section “4.11.5(1), Flowchart of enc_main”.	
Arguments	ch	: Encoder channel number 0: specify channel 0, 1: specify channel 1
Return value	0	: Normal termination
	Others	: Abnormal termination (error code of the encoder interface)

(3) fac_get_cmd

fac_get_cmd		
Synopsis	Function for acquiring the command	
Header	-	
Declaration	static uint32_t fac_get_cmd(char_t *p_arg[], const uint32_t arg_max);	
Description	This function acquires the input command.	
Arguments	p_arg	: The starting address of the RAM for storing the acquired command
	arg_max	: Maximum number of arguments
Return value	: Number of arguments	

(4) fac_cmd_single

fac_cmd_single		
Synopsis	Function for executing the single-turn data acquisition command	
Header	-	
Declaration	static void fac_cmd_single(char_t *p_arg[], const uint32_t arg_num);	
Description	This function executes the single-turn data acquisition command.	
Arguments	p_arg	: Not used
	arg_num	: Number of arguments
Return value	None	

(5) fac_cmd_multi

fac_cmd_multi	
Synopsis	Function for executing the multi-turn data acquisition command
Header	-
Declaration	static void fac_cmd_multi(char_t *p_arg[], const uint32_t arg_num);
Description	This function executes the multi-turn data acquisition command.
Arguments	p_arg : Not used arg_num : Number of arguments
Return value	None

(6) fac_cmd_encid

fac_cmd_encid	
Synopsis	Function for executing the ID acquisition command for the encoder
Header	-
Declaration	static void fac_cmd_encid(char_t *p_arg[], const uint32_t arg_num);
Description	This function executes the ID acquisition command for the encoder.
Arguments	p_arg : Not used arg_num : Number of arguments
Return value	None

(7) fac_cmd_req

fac_cmd_req	
Synopsis	Function for executing the request command
Header	-
Declaration	static void fac_cmd_req(char_t *p_arg[], const uint32_t arg_num);
Description	This function executes the request command.
Arguments	p_arg : The starting address where the command arguments array is stored arg_num : Number of arguments
Return value	None

(8) fac_cmd_e2prom_write

fac_cmd_e2prom_write	
Synopsis	Function for executing the write command for the E2PROM
Header	-
Declaration	static void fac_cmd_e2prom_write(char_t *p_arg[], const uint32_t arg_num);
Description	This function executes the write command for the E2PROM.
Arguments	p_arg : The starting address where the command arguments array is stored arg_num : Number of arguments
Return value	None

(9) fac_cmd_e2prom_read

fac_cmd_e2prom_read	
Synopsis	Function for executing the read command for the E2PROM
Header	-
Declaration	static void fac_cmd_e2prom_read(char_t *p_arg[], const uint32_t arg_num);
Description	This function executes the read command for the E2PROM.
Arguments	p_arg : The starting address where the command arguments array is stored arg_num : Number of arguments
Return value	None

(10) fac_cmd_reset_single

fac_cmd_reset_single	
Synopsis	Function for executing the single-rotation data reset command
Header	-
Declaration	static void fac_cmd_reset_single(char_t *p_arg[], const uint32_t arg_num);
Description	This function executes the single-rotation data reset command.
Argumetns	p_arg : Not used arg_num : Number of arguments
Return value	None

(11) fac_cmd_reset_multi

fac_cmd_reset_multi	
Synopsis	Function for executing the multi-turn data and all-error reset command
Header	-
Declaration	static void fac_cmd_reset_multi(char_t *p_arg[], const uint32_t arg_num);
Description	This function executes the multi-turn data and all-error reset command.
Arguments	p_arg : Not used arg_num : Number of arguments
Return value	None

(12) fac_cmd_reset_all

fac_cmd_reset_all	
Synopsis	Function for executing the all-error reset command
Header	-
Declaration	static void fac_cmd_reset_all(char_t *p_arg[], const uint32_t arg_num);
Description	This function executes the all-error reset command.
Arguments	p_arg : Not used arg_num : Number of arguments
Return value	None

(13) fac_cmd_elctimer

fac_cmd_elctimer	
Synopsis	Execution function of the ELC timer command
Header	-
Declaration	static void fac_cmd_elctimer(char_t *p_arg[], const uint32_t arg_num);
Description	Function for executing ELC timer command.
Arguments	p_arg : The starting address where the command arguments array is stored arg_num : Number of arguments
Return value	None

(14) fac_cmd_elcstop

fac_cmd_elcstop	
Synopsis	Execution function of the ELC stop command
Header	-
Declaration	static void fac_cmd_elcstop(char_t *p_arg[], const uint32_t arg_num);
Description	Function for executing ELC stop command.
Arguments	p_arg : Not used arg_num : Number of arguments
Return value	None

(15) fac_cmd_exit

fac_cmd_exit	
Synopsis	Function for executing the end command
Header	-
Declaration	static void fac_cmd_exit(char_t *p_arg[], const uint32_t arg_num);
Description	This function executes the end command.
Arguments	p_arg : Not used arg_num : Number of arguments
Return value	None

(16) fac_trans_req

fac_trans_req	
Synopsis	Function for requesting a command transfer
Header	-
Declaration	static r_fac_err_t fac_trans_req(r_fac_req_t *const p_req);
Description	This function requests a command transfer. This function is used for the commands other than the E2PROM command.
Arguments	p_req : Address where the transfer request data starts
Return value	err_code : Error code

(17) fac_trans_e2prom

fac_trans_e2prom	
Synopsis	Function for requesting transfer of the E2PROM command
Header	-
Declaration	static r_fac_err_t fac_trans_e2prom(r_fac_e2prom_data_t *const p_e2prom_data);
Description	This function requests transfer of the E2PROM command.
Arguments	p_e2prom_data : Address where the E2PROM transfer request data starts
Return value	err_code : Error code

(18) fac_trans_timer

fac_trans_timer	
Synopsis	Transfer request function for ELC timer command
Header	-
Declaration	static r_fac_err_t fac_trans_timer(r_fac_req_t *const p_req);
Description	Function for transferring request of ELC timer command.
Arguments	p_req : First address of ELC timer transfer request data
Return value	err_code : Error code

(19) fac_elc_stop

fac_elc_stop	
Synopsis	Stop function of ELC timer command
Header	-
Declaration	static r_fac_err_t fac_elc_stop(void);
Description	Stop transfer request of the ELC timer command.
Arguments	None
Return value	err_code : Error code

(20) callback_req_result

callback_req_result	
Synopsis	Callback function for notifying the results of the transmission of requests to the FA-CODER
Header	-
Declaration	static void callback_req_result(r_fac_result_t *p_result, uint8_t *p_rxdf);
Description	This function stores the results in memory and sets the acquisition-completed flag.
Arguments	p_result : Address where the result of transfer starts. See section “4.10.1(4) r_fac_result_t”. p_rxdf : Address where the received data starts
Return value	None

(21) callback_e2prom_result

callback_e2prom_result	
Synopsis	Callback function for notifying the results of the transmission from or to the E2PROM of the FA-CODER
Header	-
Declaration	static void callback_e2prom_result(r_fac_result_t *p_result, uint8_t adf, uint8_t edf);
Description	This function stores the results in memory and sets the acquisition-completed flag.
Arguments	p_result : Address where the result of transfer starts. See section “4.10.1(4) r_fac_result_t”. adf : Address field value edf : E2PROM field value
Return value	None

(22) callback_elctimer_result

callback_elctimer_result	
Synopsis	Callback function for notifying the result of request to the FA-CODER
Header	-
Declaration	static void callback_elctimer_result(r_fac_result_t *p_result, uint8_t *p_rxdf);
Description	Stores the results in memory and updates the counter.
Arguments	p_result : Address where the result of transfer starts. See section “4.10.1(4) r_fac_result_t”. p_rxdf : Address where the received data starts
Return value	None

4.11.4 Variables of Sample Program

Table 4.7 lists the main static type variables. Const type variables are not used.

Table 4.7 Main Static Type Variables

Type	Variable Name	Description	Function to be used
bool	fac_done	Result acquisition completed flag Initial value: false	fac_trans_req fac_trans_e2prom callback_req_result fac_trans_timer callback_e2prom_result
r_fac_result_t*	p_fac_result	Address where the result of the acquisition of data starts	fac_cmd_single fac_cmd_multi fac_cmd_encid fac_cmd_req fac_cmd_e2prom_write fac_cmd_e2prom_read fac_cmd_reset_single fac_cmd_reset_multi fac_cmd_reset_all callback_req_result callback_e2prom_result
uint8_t*	p_fac_rxdf	Address where the acquired data starts	fac_cmd_single fac_cmd_multi fac_cmd_encid fac_cmd_req callback_req_result
uint8_t	fac_adf	Address field value	fac_cmd_e2prom_write fac_cmd_e2prom_read callback_req_result
uint8_t	fac_edf	E2PROM field value	fac_cmd_e2prom_write fac_cmd_e2prom_read callback_req_result
bool	fac_elc_flg	ELC event input operation flag Initial value：false	fac_cmd_elctimer fac_cmd_elcstop
r_fac_result_t	fac_ti_result[]	Data acquisition result ring buffer for ELC event input operation	fac_cmd_elcstop callback_elctimer_result
uint32_t	fac_ti_single_turn[]	Acquisition data ring buffer for ELC event input operation	fac_cmd_elcstop callback_elctimer_result
uint32_t	fac_ti_count	Ring buffer storage position counter	fac_cmd_elctimer fac_cmd_elcstop callback_elctimer_result
uint32_t	fac_ti_valid	Ring buffer valid data quantity	fac_cmd_elctimer fac_cmd_elcstop callback_elctimer_result
bool	fac_ti_full	Ring buffer full flag	fac_cmd_elctimer callback_elctimer_result

4.11.5 Flowchart of Main Processing

(1) Flowchart of enc_main

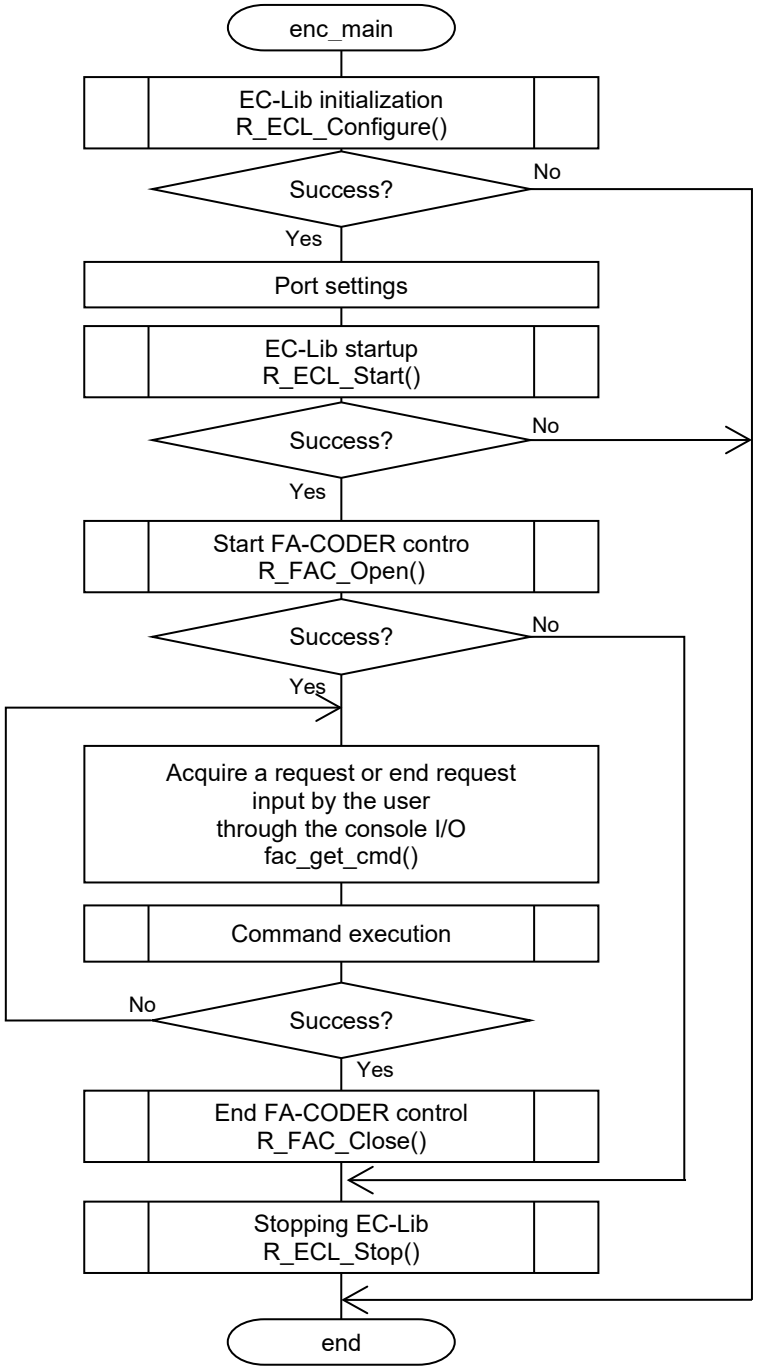


Figure 4.3 Flowchart of enc_main Function

(2) Flowchart of fac_get_cmd

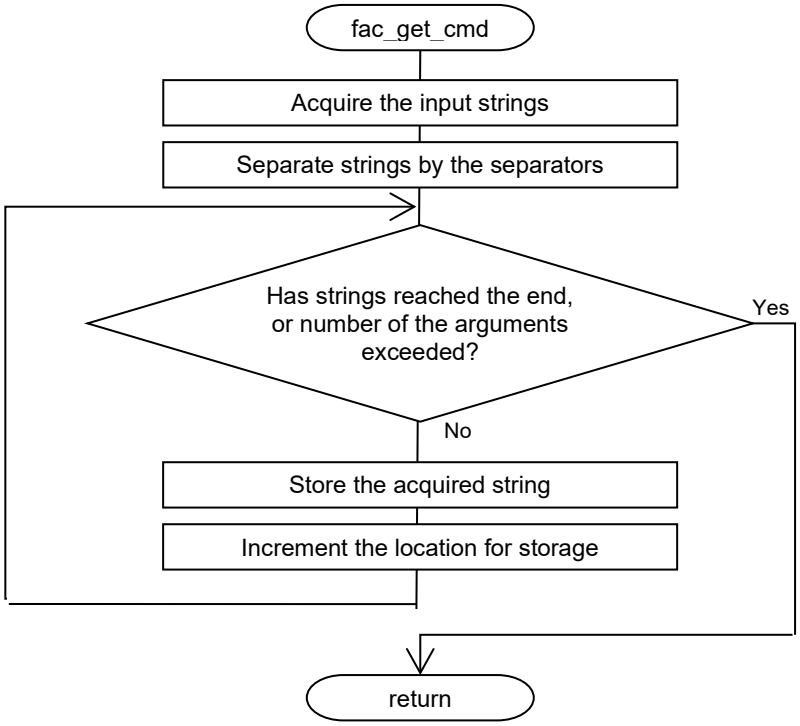


Figure 4.4 Flowchart of fac_get_cmd Function

(3) Flowchart of fac_cmd_xxx

This section shows the flowchart of the functions described in section “4.11.3(4), fac_cmd_single” to section “4.11.3(6), fac_cmd_encid”.

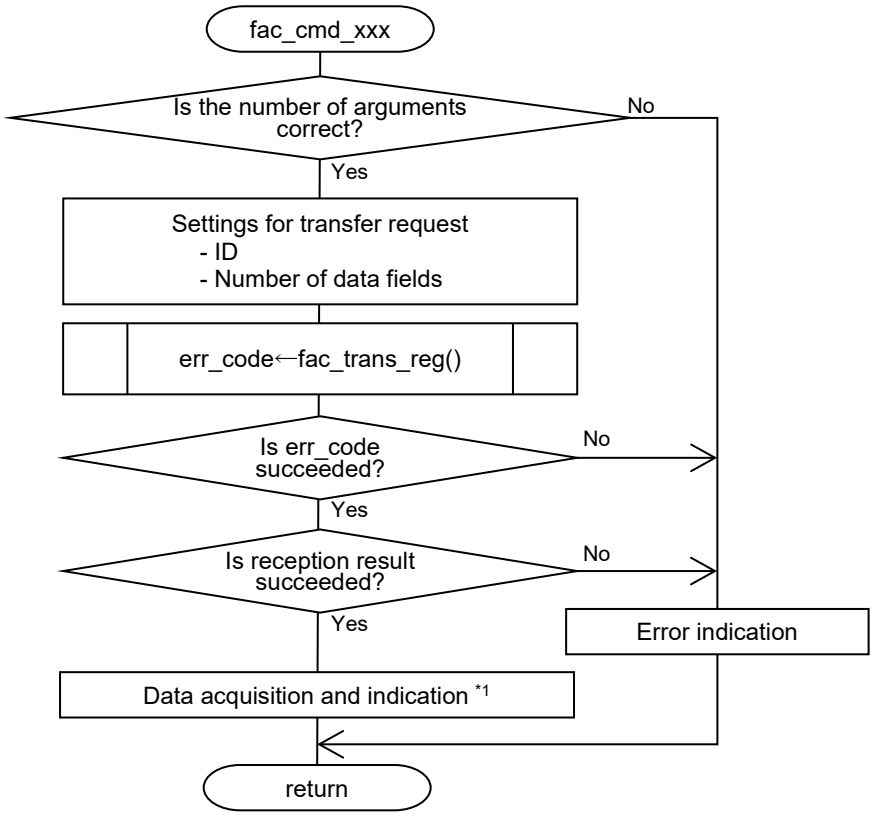


Figure 4.5 Flowchart of fac_cmd_xxx Functions

Note: 1. The single-turn data acquisition command acquires the absolute values within a single rotation. The multi-turn acquisition command acquires data for multiple rotations. The encoder ID acquisition command acquires the encoder ID.

(4) Flowchart of fac_cmd_req

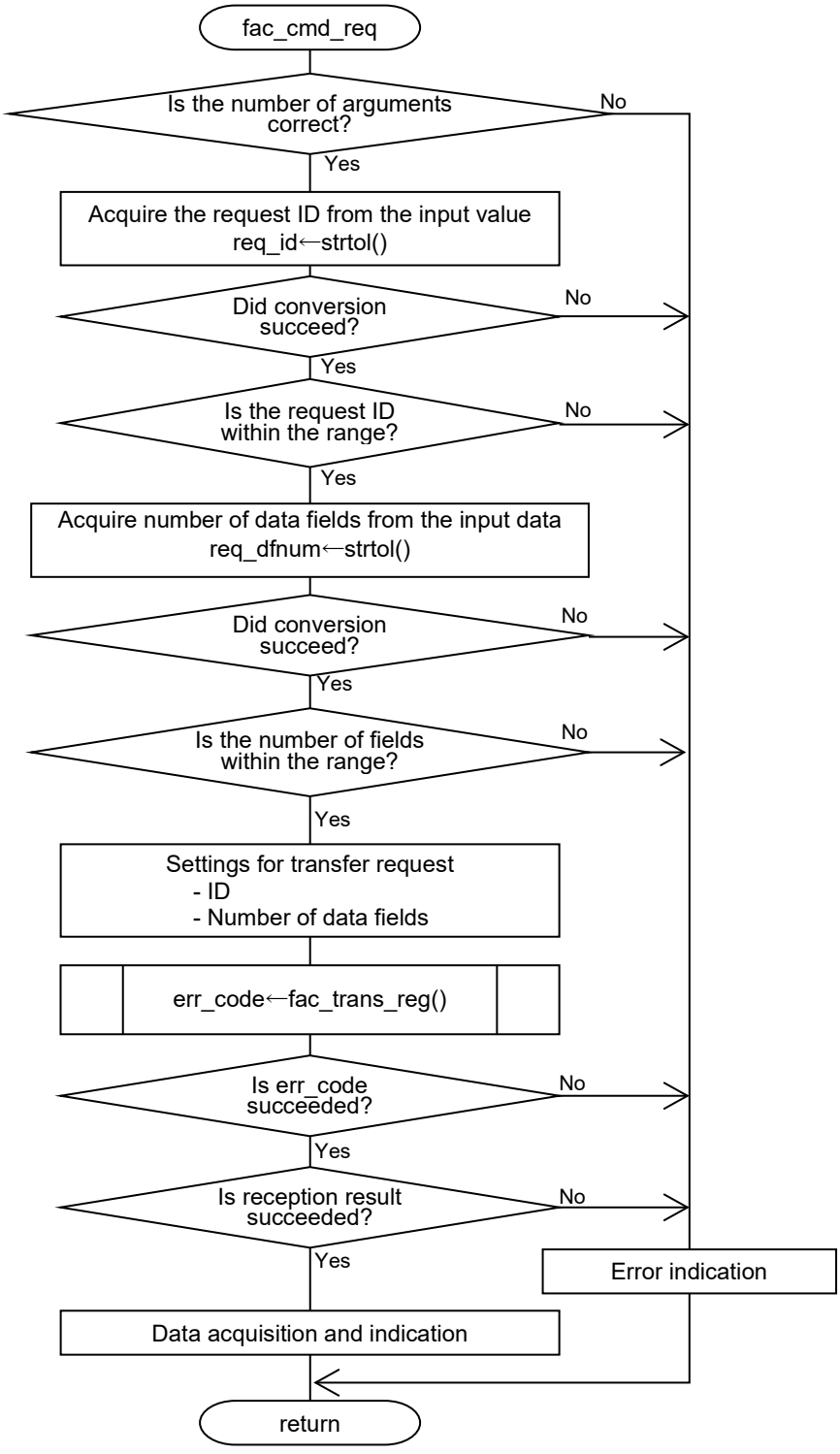


Figure 4.6 Flowchart of fac_cmd_req Function

(5) Flowchart of fac_cmd_e2prom_write

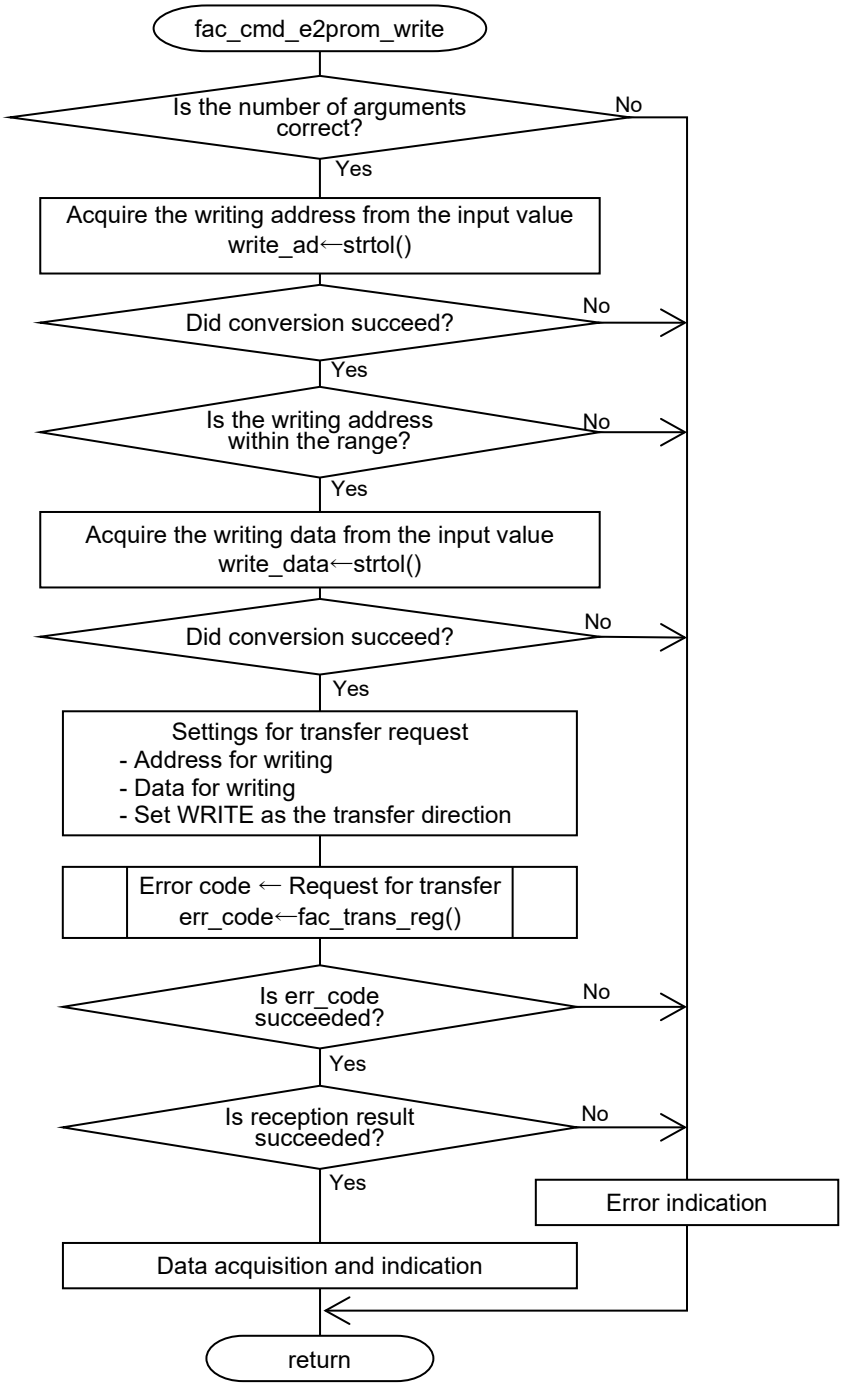


Figure 4.7 Flowchart of fac_cmd_e2prom_write Function

(6) Flowchart of fac_cmd_e2prom_read

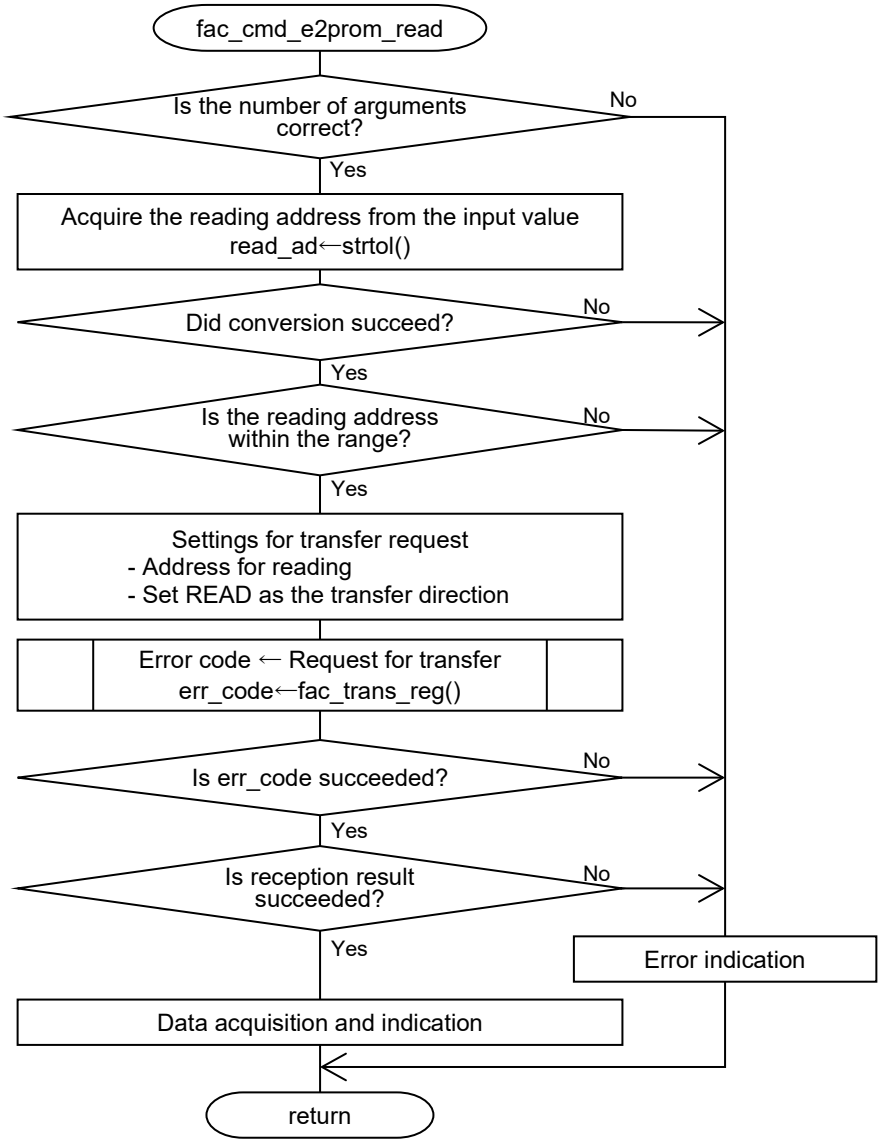


Figure 4.8 Flowchart of fac_cmd_e2prom_read Function

(7) Flowchart of fac_cmd_reset_xxx

This section shows the flowchart of the functions described in section “4.11.3(10), fac_cmd_reset_single” to section “4.11.3(12), fac_cmd_reset_all”.

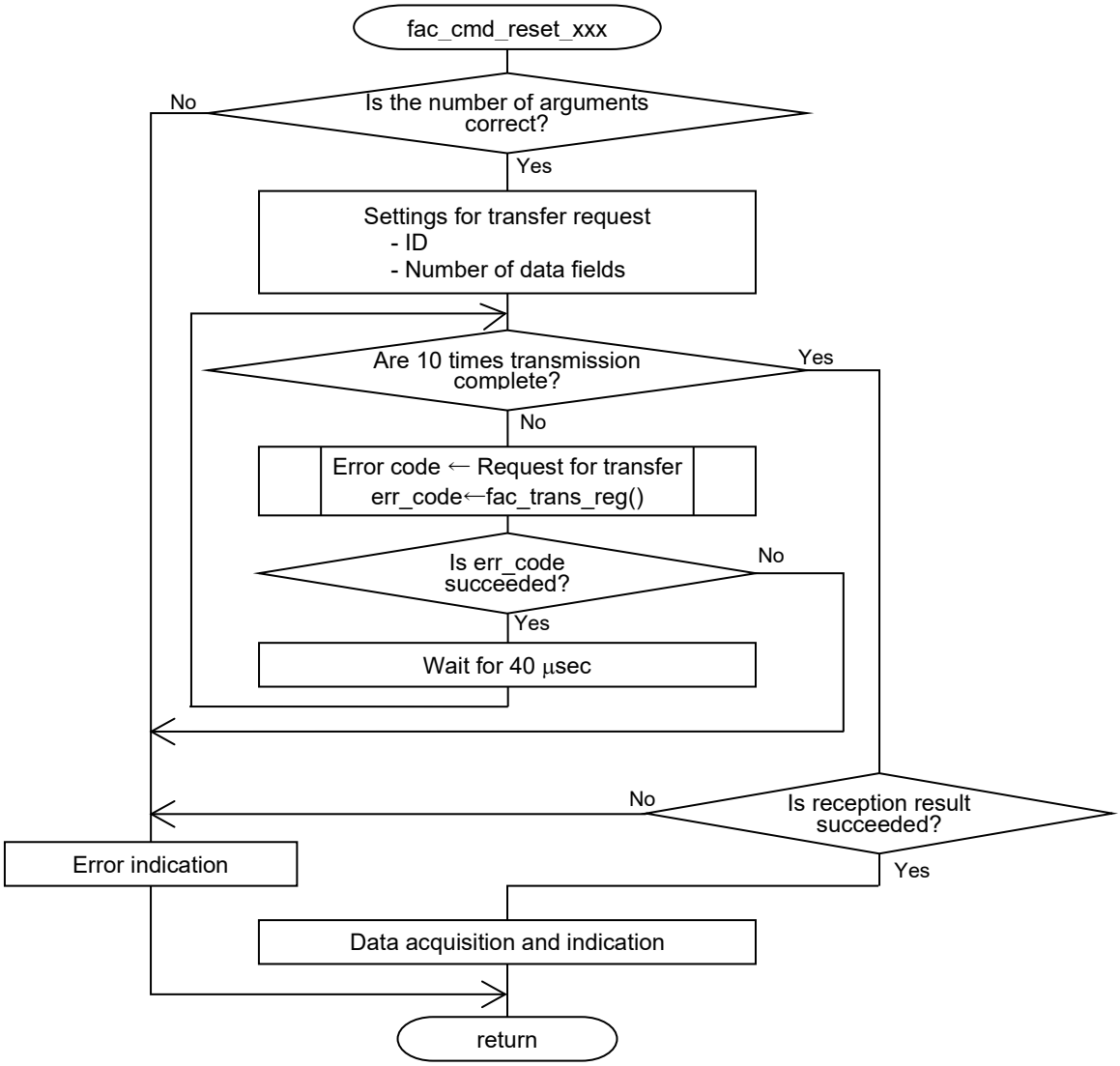


Figure 4.9 Flowchart of fac_cmd_reset_xxx Function

(8) Flowchart of fac_cmd_elctimer

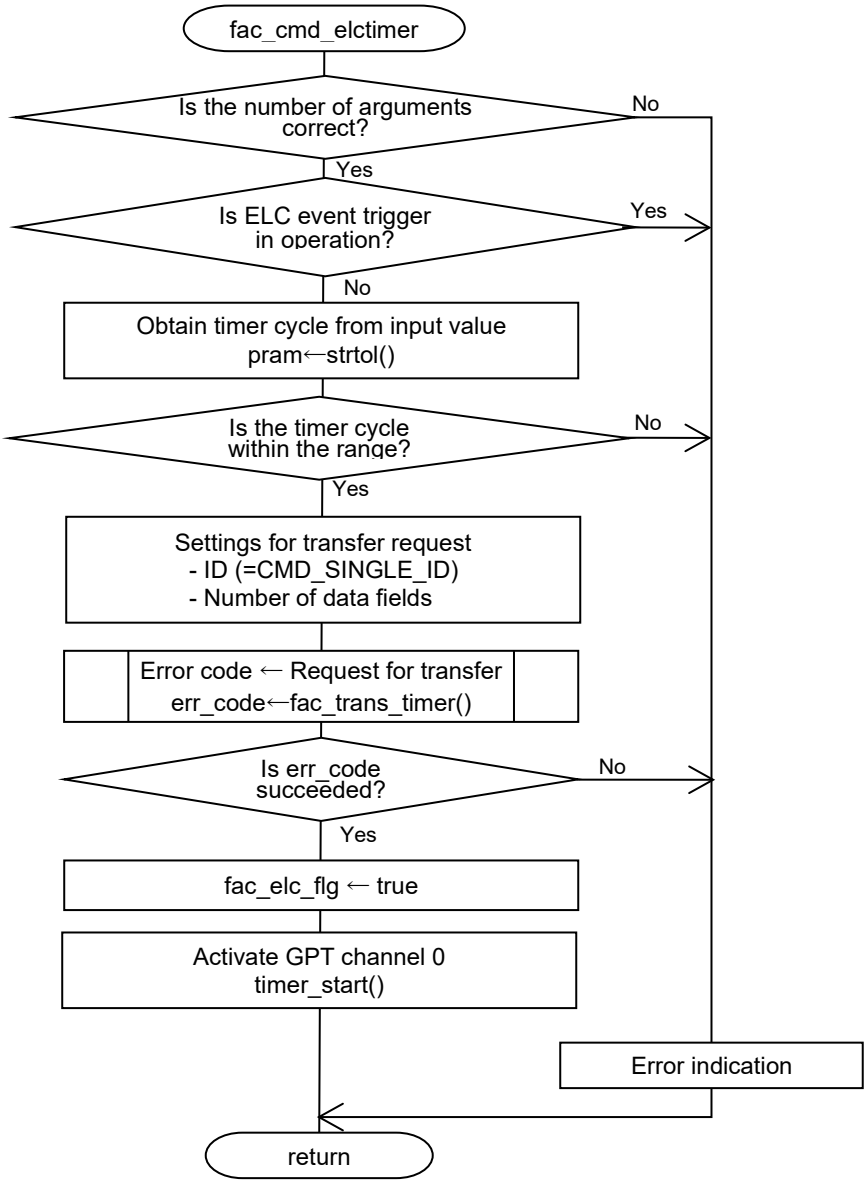


Figure 4.10 Flowchart of fac_cmd_elctimer function

(9) Flowchart of fac_cmd_elcstop

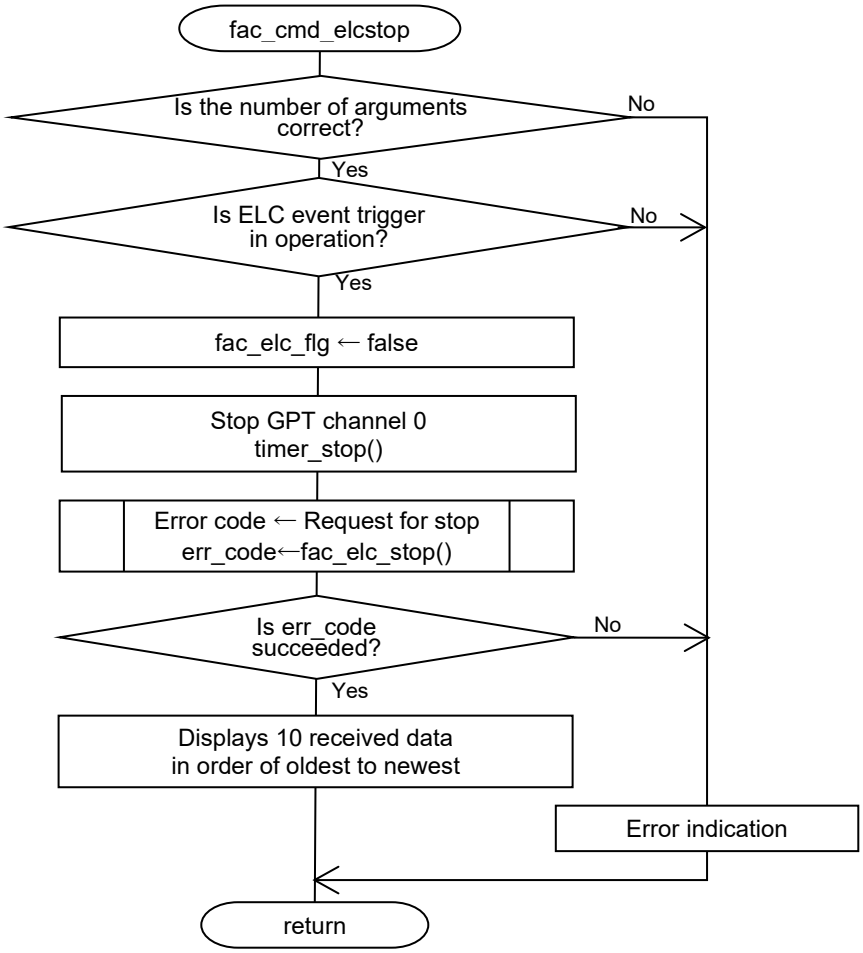


Figure 4.11 Flowchart of fac_cmd_elcstop function

(10) Flowchart of fac_cmd_exit

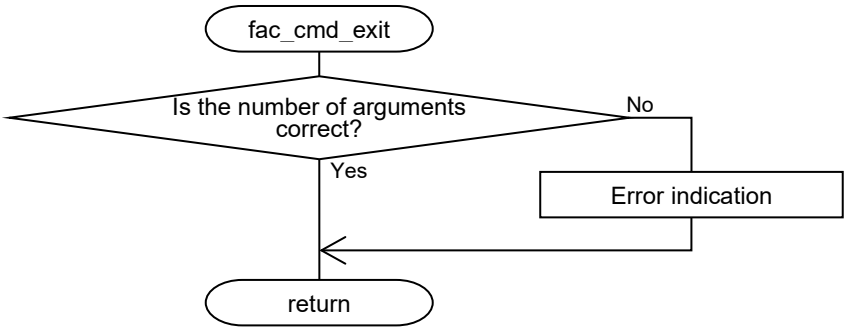


Figure 4.12 Flowchart of fac_cmd_exit Function

(11) Flowchart of fac_trans_xxx

This section shows the flowchart of the functions described in section “4.11.3(16), fac_trans_req” and section “4.11.3(17), fac_trans_e2prom”.

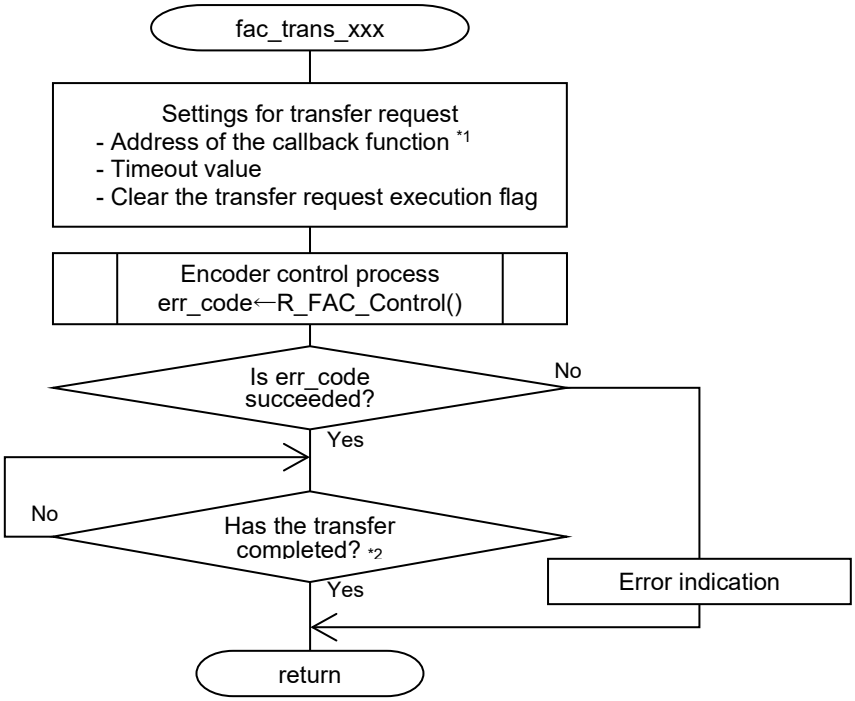


Figure 4.13 Flowchart of fac_trans_xxx Functions

Note: 1. The fac_trans_req function sets callback_req_result. The fac_cmd_e2prom_write and the fac_cmd_e2prom_read functions set callback_e2prom_result.
2. Transfer completion flag is set within a callback function.

(12) Flowchart of fac_trans_timer

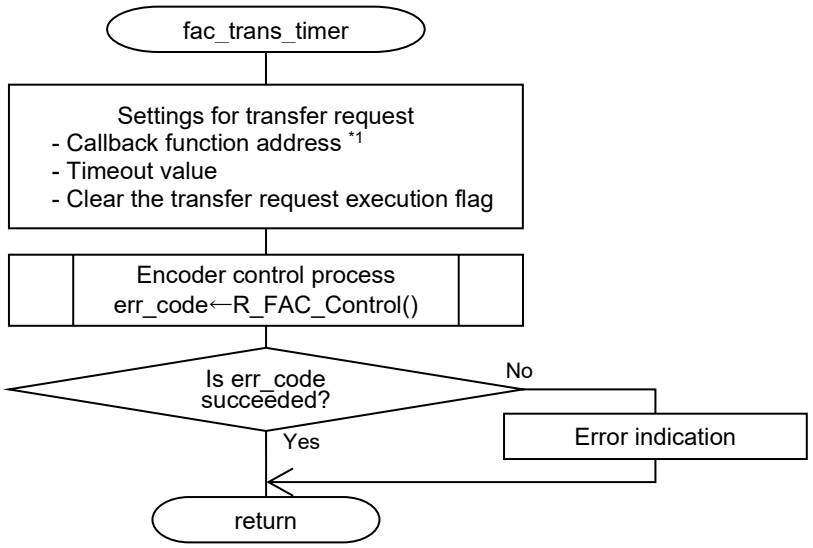


Figure 4.14 Flowchart of fac_trans_timer function

Note: 1. The fac_trans_timer function sets callback_elctimer_result as the callback function.

(13) Flowchart of fac_elc_stop

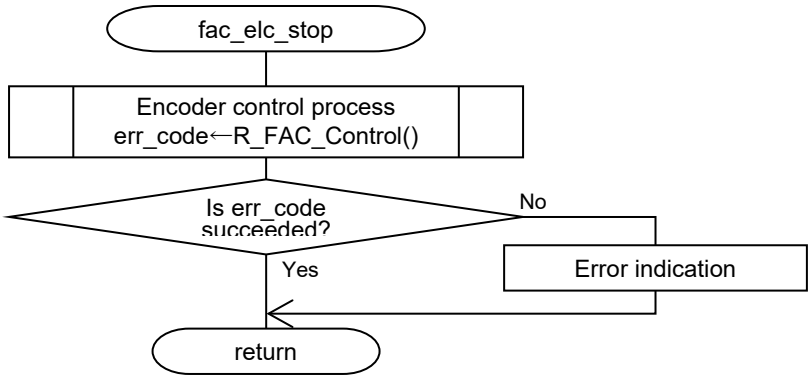


Figure 4.15 Flowchart of fac_elc_stop function

(14) Flowchart of callback_xxx_result

This section shows the flowchart of the functions described in section “4.11.3(20), callback_req_result” and section “4.11.3(21), callback_e2prom_result”.

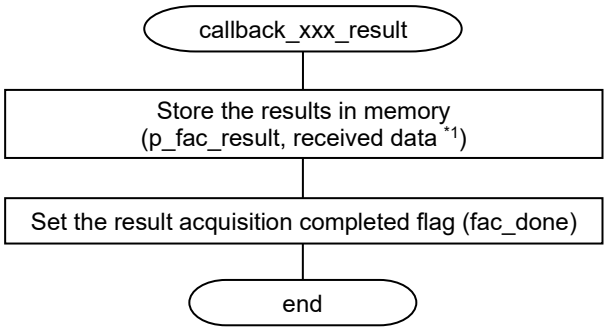


Figure 4.16 Flowchart of callback_xxx_result Functions

Note: 1. The callback_req_result function stores p_fac_rxdf (received data field). The callback_e2prom_result function stores data in fac_adf (ADF register) and fac_edf (EDF register).

(15) Flowchart of callback_elctimer_result

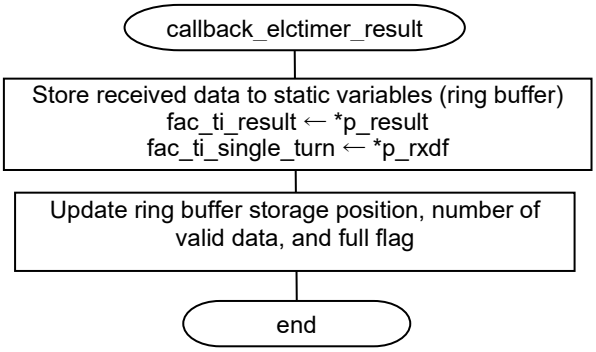


Figure 4.17 Flowchart of callback_elctimer_result Function

4.11.6 Operation Sequence

(1) Startup Sequence

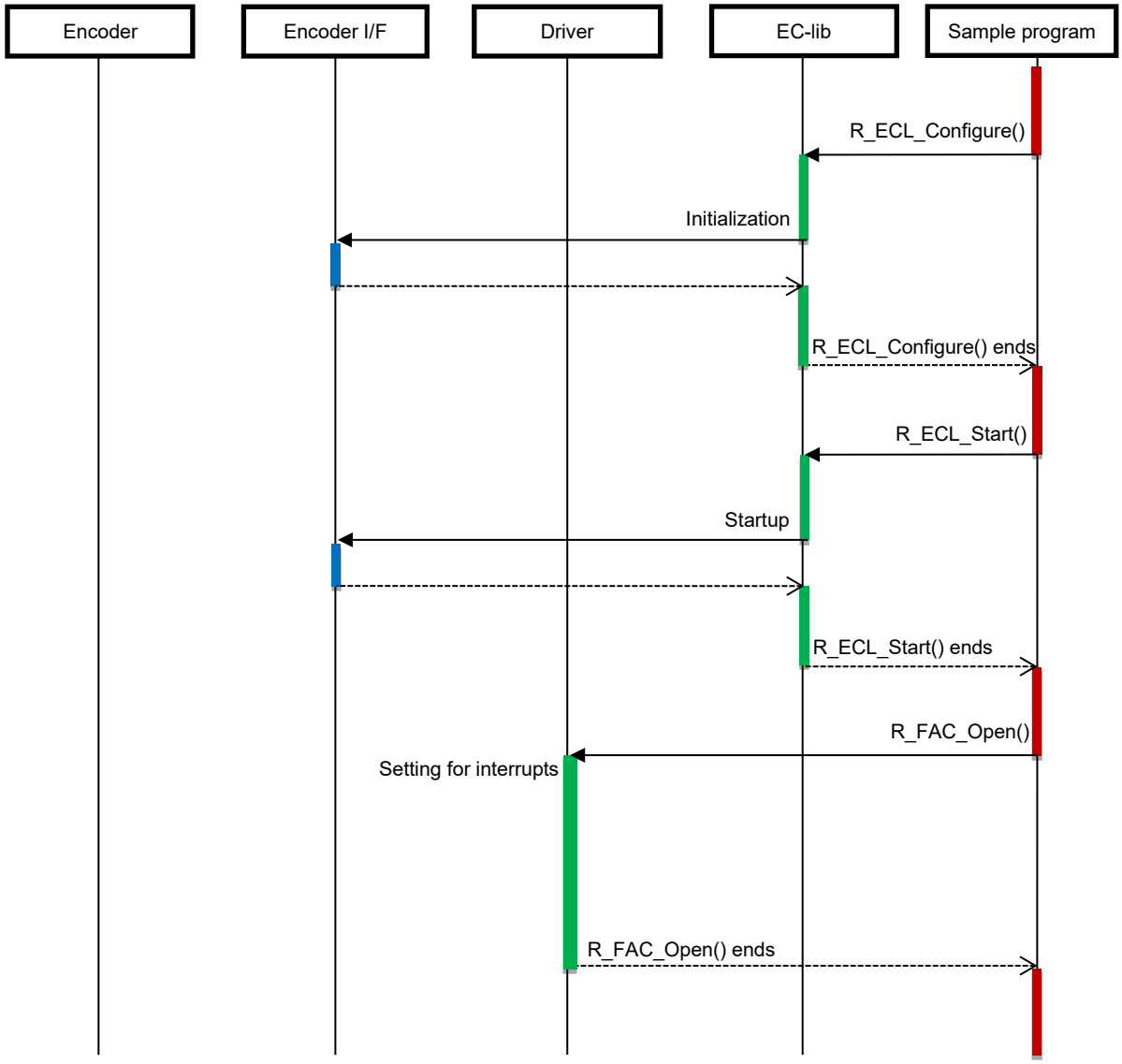


Figure 4.18 Startup Sequence Diagram

(2) Request Data Reception Sequence

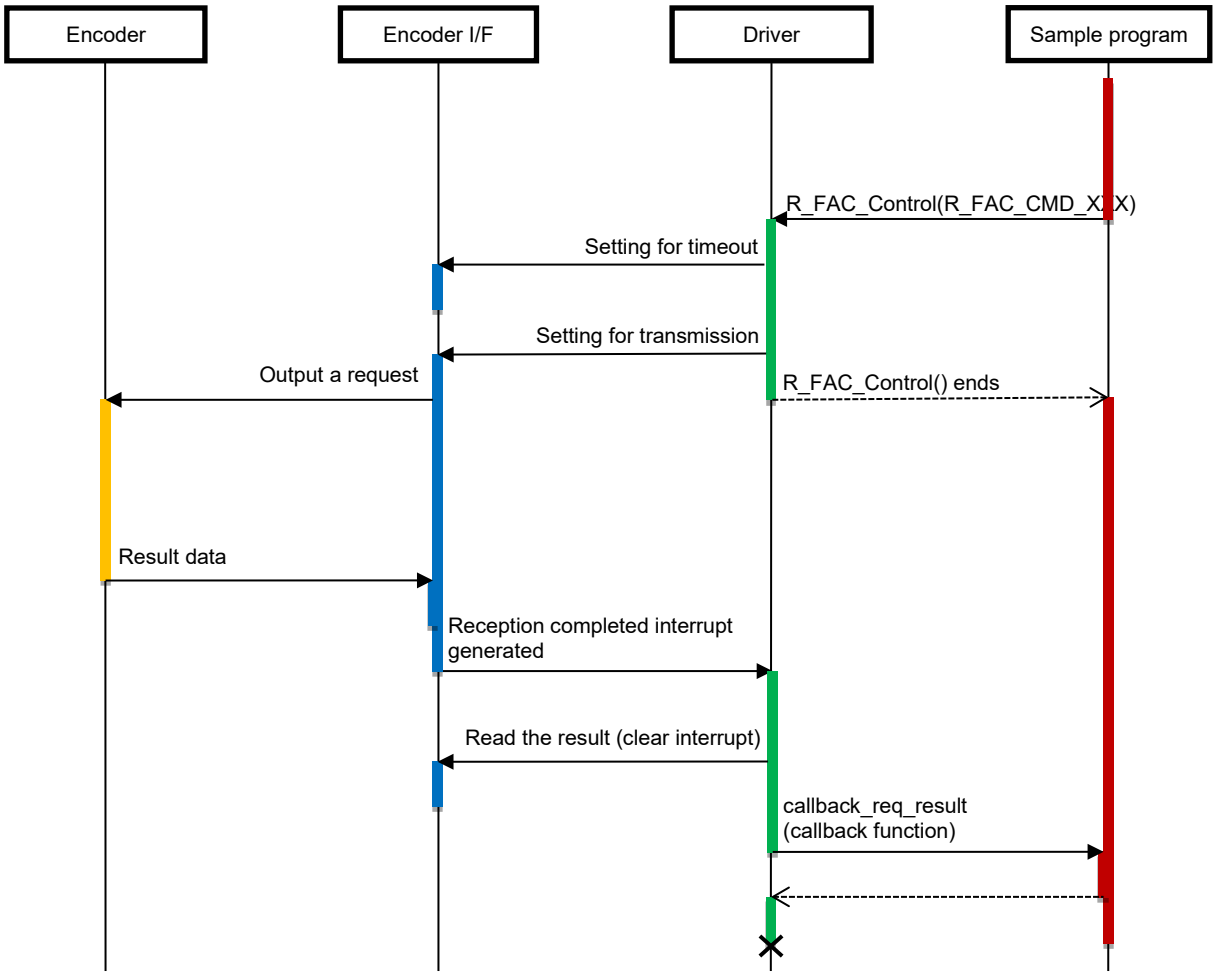
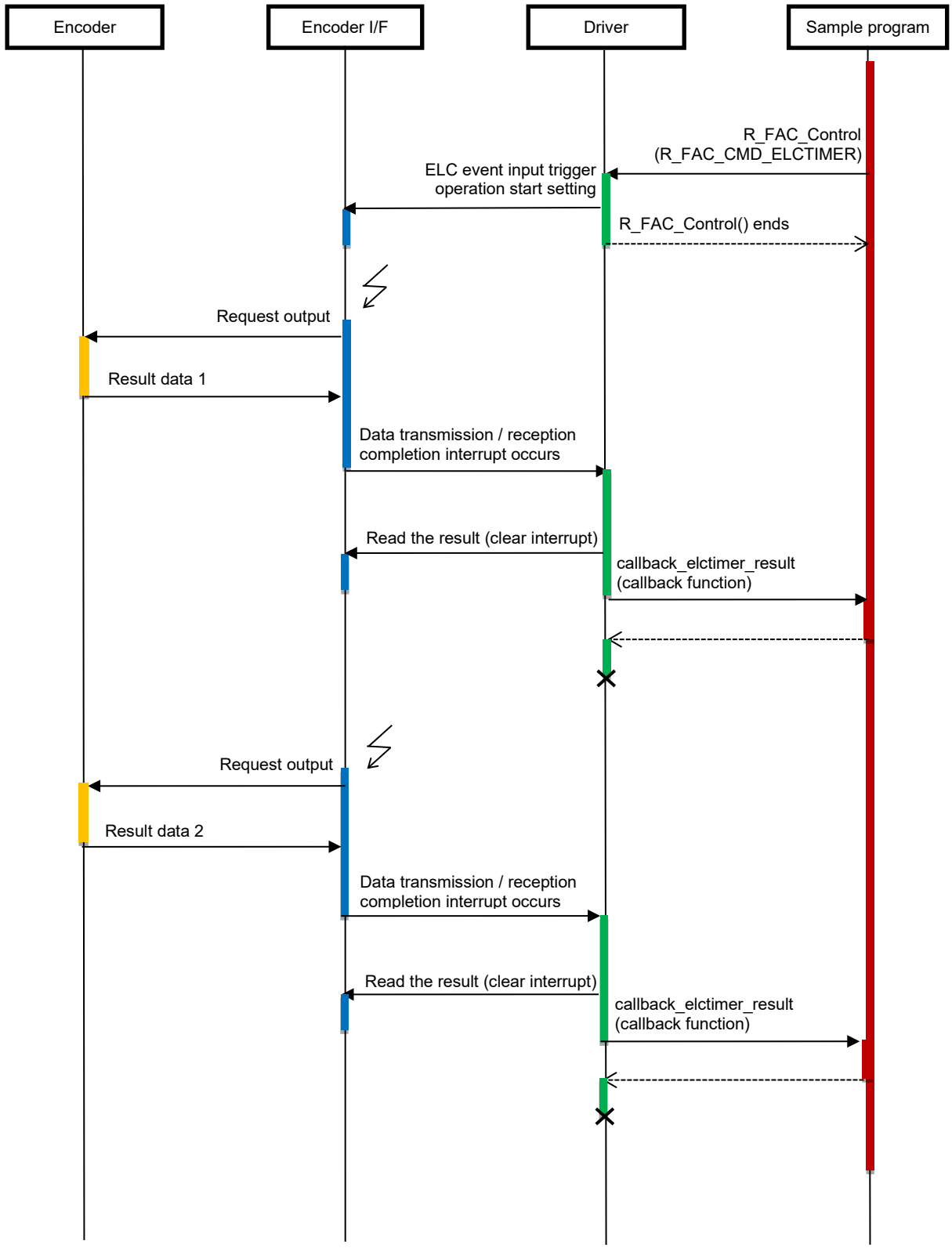


Figure 4.19 Request Data Reception Sequence Diagram

(3) ELC Event Input Trigger Operation Sequence



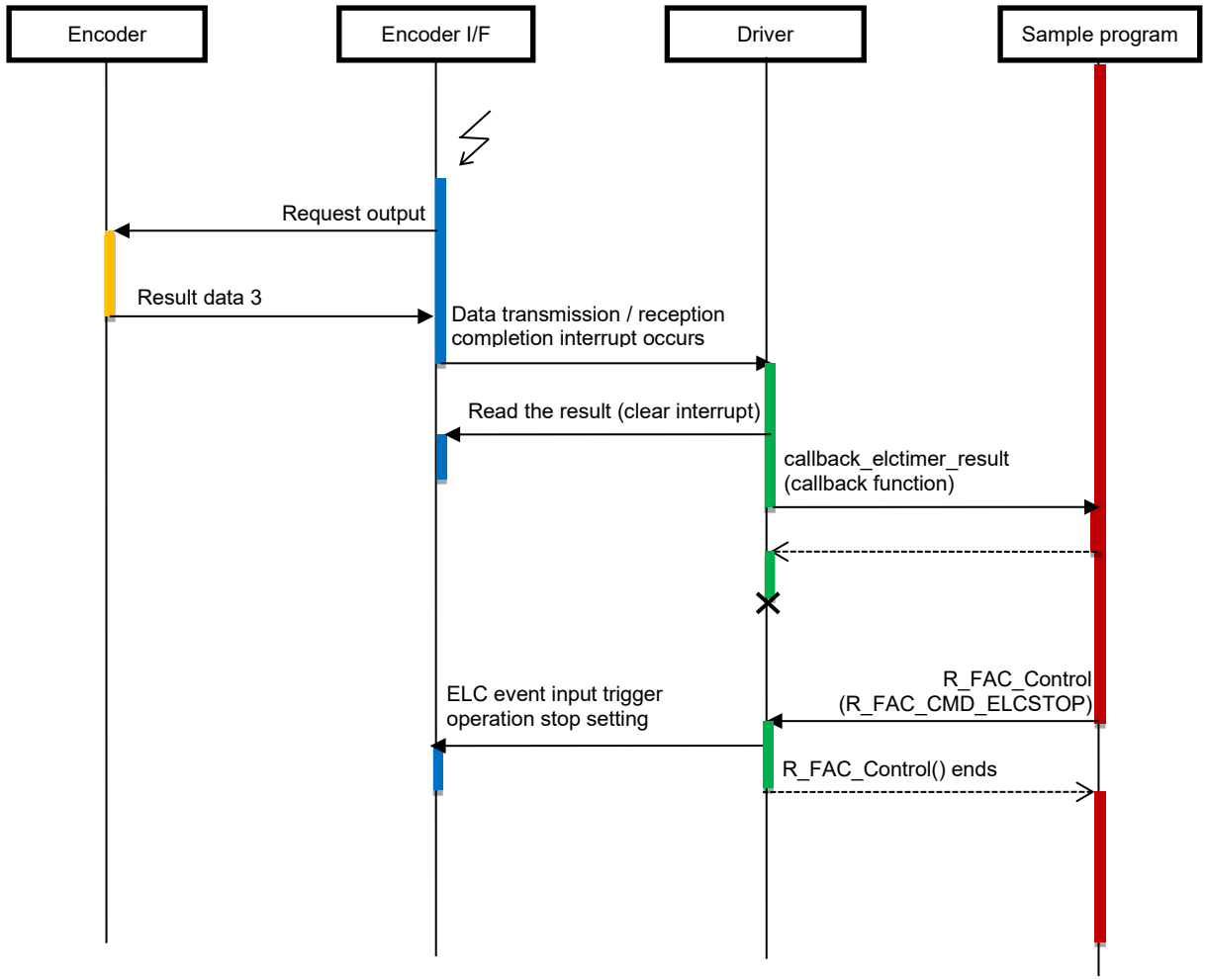


Figure 4.20 ELC Event Input Trigger Operation Sequence Diagram

(4) Stop Sequence

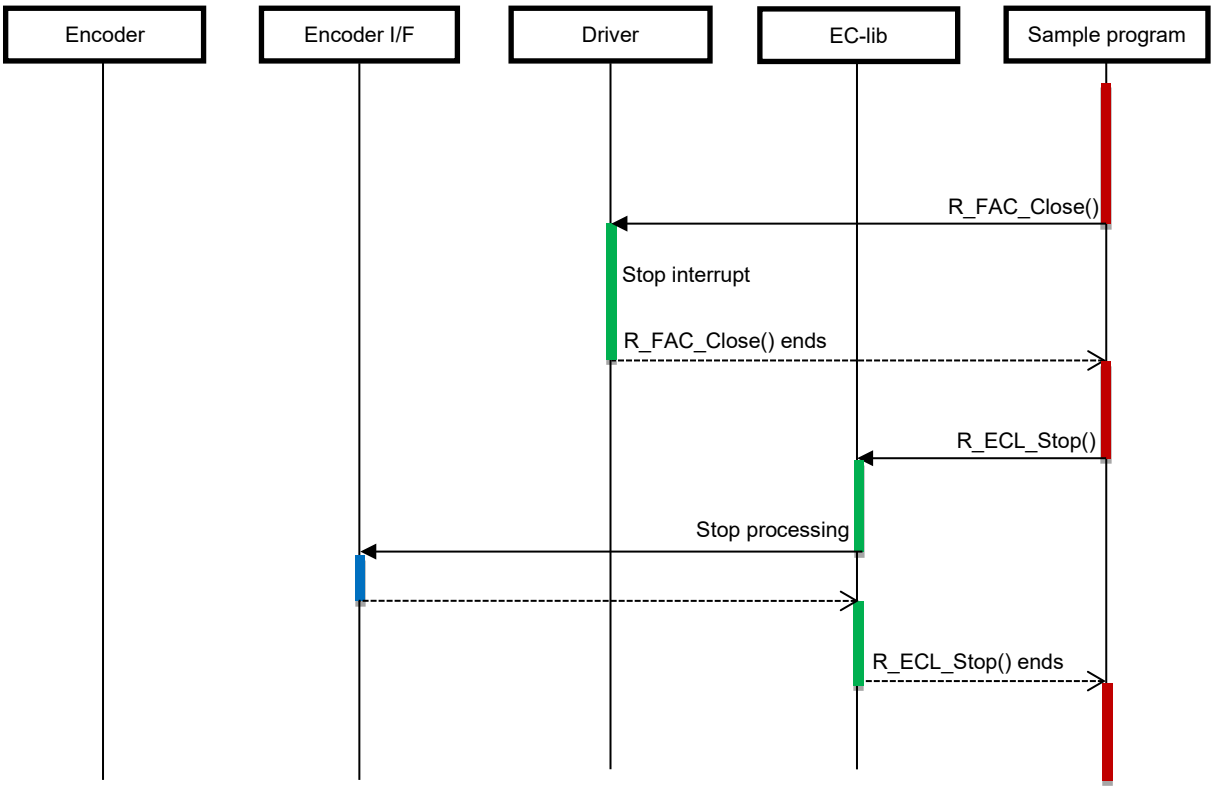


Figure 4.21 Stop Sequence Diagram

4.11.7 Console Commands

This sample program supports the positional encoder from Tamagawa Seiki (FA-CODER) “TS5667N120” and “TS5702N142”. The commands available for input from the console are listed below.

Table 4.8 Console Commands

Command	Description
single	Acquires single-turn data. *1
multi	Acquires multi-turn data. *1
encid	Acquires the encoder ID. *1
write *x y*	Writes data. *1 This is executed by specifying the write address *x* and data *y* (0 to 255). *2
read *x*	Reads data. *1 This is executed by specifying the read address *x*. *2
req *x y*	Executing a request Specify request ID *x* and data field count *y* to execute the request. For specific settings of request IDs and data field counts,refer to “Table 4.9 List of Arguments for the ‘req’ Command”.
reset_single	Resets data for a single rotation. *1
reset_multi	Resets data for multiple rotations and all errors. *1
reset_all	Resets all errors. *1
elctimer *val*	Obtain single-turn data in a timer cycle as a timer event synchronized operation. *1 Specify timer cycle *val* in us (max. 6990 us) for execution.
elcstop	Timer event synchronized operation is terminated and acquired data is displayed.
exit	End of Program

Note: 1. Command for encoders “TS5667N120” and “TS5702N142”.
2. Available range of read / write address x is 0 to 79 for “TS5667N120”, 0 to 127 for “TS5702N142”.

Table 4.9 List of Arguments for the ‘req’ Command

Request ID	Data Field Count	Description
0	3	Acquires single-turn data. *1
1	3	Acquires multi-turn data. *1
2	1	Acquires the encoder ID. *1
3	8	Acquires single-turn data, encoder ID, multi-turn data, and encoder error information. *1
7	3	Resets all errors. *1
8	3	Resets data for a single rotation. *1
C	3	Resets data for multiple rotations and all errors. *1

Note: 1. Command for encoders “TS5667N120” and “TS5702N142”.

(1) Result of Running

After running, it will display the command prompt following the version. Please enter commands after ‘tamagawa >’ appears.

```
Tama sample program start
R_FAC_GetVersion = 4.0


tamagawa >
```

(2) Example of Command Execution

This is an example of executing the 'single' command. Based on the response from the encoder, single-turn data, request ID, alarm information, and other details are displayed.

```
tamagawa >single
single command
  result:success
    single turn data:                764010
    request id:                       0H
    parity bit(request id):           0H
    information code:                 0H
    encoder alarm:                    2H
    communication alarm:              0H
    crc data:                        EBH
tamagawa >
```

5. Sample Code

The sample code is available from the Renesas Electronics website.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr.08.22		First Edition issued
1.10	Jun.28.22	1,4-8, 20,48	Changed "Multi-Protocol Encoder I/F" notation to "Encoder I/F" Corrected the description of operating voltage Corrected the description of the device used Corrected the list of used pins Deleted memory size description
1.20	Aug.26.22	18	Remove form error (fome) from result status
1.30	Apr.14.23	7,15,16,28 ,47	Add argument p_info to the function R_FAC_Open(). Add bit rate indices Table 4.4. Add description for r_fac_info_t structure. Correct title of Table 4.7. Update notes for Table 4.8 console commands.
1.40	Apr 22.24	4 15 16	Update description of board name. Remove description about location of integer type definition. Add element of reception guard band rx_guard in the structure for initialization information r_fac_info_t.
3.00	Oct 17.25	1, 3, 4 5 7 to 12 27 to 38 44 45	Change description for trademarks. Update description of the used pins. Revise to unify description for functions. Revise figures of flowchart. Revise Table 4.8 and add Table 4.9. Add example of command execution.
4.00	Mar 6.26	7-38	Change the prefix of pointer variables to “p_”. (ex. pinfo -> p_info)

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/.