
RZ/T2M Group

Dual Encoder Sample Program

Summary

This application note explains a sample program that simultaneously connects two different encoders, one compliant with the A-format™ communication protocol specification (“A-format™ Specification”) and the other with the EnDat 2.2 Specification. This sample program acquires and displays information for each by using the RZ/T2M Encoder I/F Configuration Library.

The major features of the program are listed below.

- Simultaneous connection of encoders compliant with the A-format™ specification and encoders compliant with the EnDat 2.2 specification.
- Supports A-format™ command codes.
- Obtains angle information, etc. from an encoder conforming to the A-format™ specification (MAR-M50A or SAR-HL700A from Nikon).
- Supports the mode command and the MRS codes used in EnDat 2.2.
- Obtains angle information, etc. from an encoder (EQN1035 from HEIDENHAIN) compliant with EnDat 2.2 specifications.

Target Device

RZ/T2M

When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation of the modified program.

Table of Contents

1. Specifications	4
2. Operating Environment.....	5
3. Peripheral Modules.....	6
3.1 Pins.....	6
4. Software	7
4.1 A-format Driver Function	7
4.2 EnDat Driver Function	7
4.3 File Structures	7
4.4 Functions	8
4.5 Specifications of A-format API Functions	9
4.5.1 R_A_AS_Open	9
4.5.2 R_A_AS_Close.....	9
4.5.3 R_A_AS_GetVersion.....	10
4.5.4 R_A_AS_Control	10
4.5.5 A-format Control Commands.....	11
4.6 Specification of A-format User-defined Functions.....	14
4.6.1 a_as_txerr_callback	14
4.6.2 a_as_rxset_callback.....	15
4.6.3 a_as_rxend_callback.....	16
4.6.4 a_as_elctimer_callback.....	17
4.7 A-format Interrupt Handler.....	18
4.7.1 a_as_ch0_int_isr	18
4.7.2 a_as_ch1_int_isr	18
4.8 Specifications of EnDat API Functions.....	19
4.8.1 R_ENDAT_Open	19
4.8.2 R_ENDAT_Close.....	19
4.8.3 R_ENDAT_GetVersion.....	20
4.8.4 R_ENDAT_Control	20
4.8.5 EnDat Control Commands	21
4.9 Specifications of EnDat User-defined Functions.....	22
4.9.1 enc_init_reset_wait_callback.....	22
4.9.2 enc_init_mem_wait_callback.....	23
4.9.3 enc_init_pram_wait_callback	23
4.9.4 enc_init_cable_wait_callback.....	24
4.9.5 endat_callback.....	24
4.9.6 endat_poscon_callback.....	25
4.9.7 endat_rdst_callback	25
4.10 EnDat Interrupt Handler	25

4.10.1 endat_ch0_int_isr	25
4.10.2 endat_ch1_int_isr	26
4.11 Interrupts	26
4.12 Constant and Error Code	27
4.13 Fixed-width Integer Types	34
4.14 Structures, Unions, and Enumerated Types	35
4.14.1 A-format Structures	35
4.14.2 A-format Unions.....	38
4.14.3 A-format Enumerated Types	39
4.14.4 EnDat Structures	40
4.14.5 EnDat Unions	44
4.14.6 EnDat Enumerated Types	45
4.15 Description of the Sample Program	46
4.15.1 Operation Outline	46
4.15.2 Sample Program Functions.....	48
4.15.3 Specifications of Sample Program Functions	49
4.15.4 Variables Used in the A-format Sample Program	57
4.15.5 Constants Used in the A-format Sample Program	57
4.15.6 Variables of EnDat Sample Program	58
4.15.7 Constants of EnDat Sample Program	59
4.15.8 Flowchart of Main Processing	60
4.15.9 Dual Encoder Operation Sequence.....	75
4.15.10 Console Commands	84
4.15.11 Procedure of Channel Switching	87
5. Sample Code.....	88
Revision History	89

1. Specifications

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

Table 1.1 Peripheral Modules and Applications

Peripheral Module	Application
Multi-Protocol Encoder I/F	RZ/T2M has 2 channels of encoder interface. The encoder interface communicates the respective encoder by loading configuration data according to the A-format specification and EnDat 2.2 specification.
Interrupt controller (ICU)	Multi-Protocol Encoder I/F interrupt control
General PWM Timer (GPT)	Generation of event cycles to be input to ELC
Event Link Controller (ELC)	Link events output by GPT to Multi-Protocol 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.

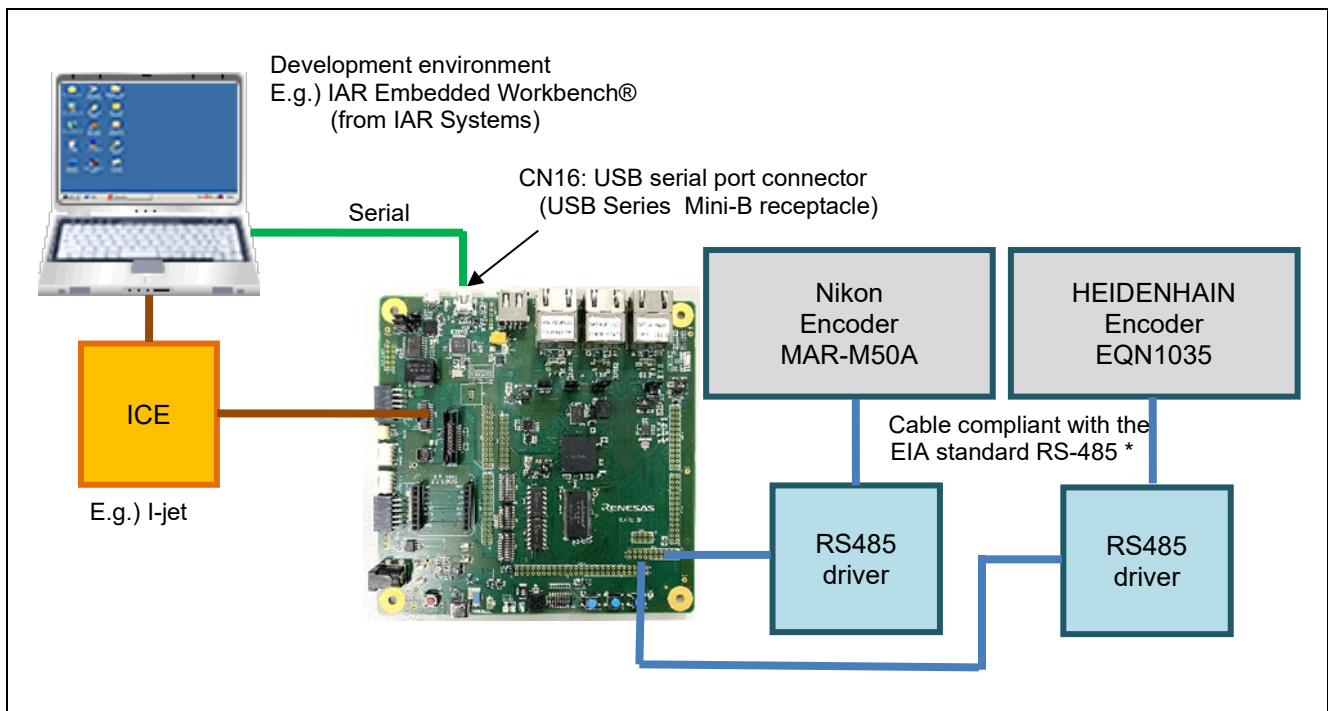


Figure 1.1 Operating Environment

Note: Contact the encoder manufacturer for the cable length that can transmit and receive.

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	1.1 V (Core) / 1.8 V (PLL, etc.) / 3.3 V (I/O)
Integrated development environment *	IAR Systems: IAR Embedded Workbench® for Arm®
	RENESAS: e ² studio
Board	RSK+RZT2M (RTK9RZT2M0C00000BE)
Devices (function to be used on the board)	None

Note Refer to the RZ/T2M Group Encoder I/F Dual Encoder 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 Used Pins and Their Functions

Channel	Port Name	I/O Port	Input/ Output	Description
A_AS0	SD0 (ENCIF0)	P01_6	Input	Data input pin
	CMND0 (ENCIF2)	P02_0	Output	Data output pin
	D_R0 (ENCIF3)	P02_2	Output	Drive/receive control pin
ENDAT_CH1	DATA_RC1 (ENCIF5)	P17_3	Input	Data input pin
	DATA_DV1 (ENCIF7)	P17_5	Output	Data output pin
	DE1 (ENCIF8)	P03_0	Output	Drive/receive control pin
	TCLK1 (ENCIF9)	P03_3	Output	Clock output pin

Table 3.2 Used Pins and Their Functions (Channel Switching Case) *

Channel	Port Name	I/O Port	Input/ Output	Description
A_AS1	SD1 (ENCIF5)	P17_3	Input	Data input pin
	CMND1 (ENCIF7)	P17_5	Output	Data output pin
	D_R1 (ENCIF8)	P03_0	Output	Drive/receive control pin
ENDAT_CH0	DATA_RC0 (ENCIF0)	P01_6	Input	Data input pin
	DATA_DV0 (ENCIF2)	P02_0	Output	Data output pin
	DE0 (ENCIF3)	P02_2	Output	Drive/receive control pin
	TCLK0 (ENCIF4)	P02_3	Output	Clock output pin

Note: How to switch A-format channel and EnDat 2.2 channel is written in “4.15.11 Procedure of Channel Switching”.

4. Software

4.1 A-format Driver Function

The functions of the A-format driver are listed below.

- 1 Initial settings
- 2 Transmission of command codes
- 3 Acquisition of reception data

4.2 EnDat Driver Function

The functions of the EnDat driver are listed below.

- 1) Initial settings
 - A) Settings of the noise filter
 - B) Initialization of the encoder (Encoder with battery unit is not supported)
 - C) Settings of propagation delay compensation
- 2) Transmission of the following request information
 - A) Mode command
 - B) MRS Code
 - C) Parameters
- 3) Reception of the encoder data
 - A) Positional value
 - B) Parameters
 - C) Additional information *

Note: In this document, "additional information" represents the additional data 1 and 2. For details, see the "EnDat Specification" which is available from HEIDENHAIN GmbH on request.

4.3 File Structures

For the file structure, refer to the release note for the RZ/T2M Group Encoder I/F Dual Encoder sample program.

4.4 Functions

Table 4.1 lists the functions for A-format. Table 4.2 lists the functions for EnDat.

Table 4.1 A-format Functions

Category	Function Name	Page Number
A-format driver API function	R_A_AS_Open	9
	R_A_AS_Close	9
	R_A_AS_GetVersion	10
	R_A_AS_Control	10
User-defined functions	a_as_txerr_callback	14
	a_as_rxset_callback	15
	a_as_rxend_callback	16
	a_as_elctimer_callback	17
Interrupt-handlers	a_as_ch0_int_isr	18
	a_as_ch1_int_isr	18

Table 4.2 EnDat Functions

Category	Function Name	Page Number
EnDat driver API functions	R_ENDAT_Open	19
	R_ENDAT_Close	19
	R_ENDAT_GetVersion	20
	R_ENDAT_Control	20
User-defined functions	enc_init_reset_wait_callback	22
	enc_init_mem_wait_callback	23
	enc_init_pram_wait_callback	23
	enc_init_cable_wait_callback	24
	endat_callback	24
	endat_poscon_callback	25
	endat_rdst_callback	25
Interrupt-handlers	endat_ch0_int_isr	25
	endat_ch1_int_isr	26

4.5 Specifications of A-format API Functions

4.5.1 R_A_AS_Open

R_A_AS_Open							
Synopsis	Starts controlling operation of the encoder.						
Header	r_a_as_rzt2_if.h						
Declaration	int32_t R_A_AS_Open(const int32_t id, r_a_as_info_t* p_info);						
Description	This function is used for the following initial settings of the A_AS module. <ol style="list-style-type: none"> 1. Setting of the noise filter 2. Setting of the communication parameters(for T2,T3,T4,T5,T9,IFMG,BR registers) 3. Clearing of state information (SS register) 4. Setting for enabling interrupts by using the INTE register. 						
Argument	id : Designates the ID code to be used. (It is defined in r_a_as_rzt2_dat.h.) <table border="0" style="margin-left: 20px;"> <tr> <td>R_A_AS0_ID</td> <td>: Specifies channel 0.</td> </tr> <tr> <td>R_A_AS1_ID</td> <td>: Specifies channel 1.</td> </tr> <tr> <td>Others</td> <td>: Setting is not allowed.</td> </tr> </table> <p>p_info : Sets information about the encoder. Designate the address of the structure r_a_as_info_t where encoder information is stored.</p>	R_A_AS0_ID	: Specifies channel 0.	R_A_AS1_ID	: Specifies channel 1.	Others	: Setting is not allowed.
R_A_AS0_ID	: Specifies channel 0.						
R_A_AS1_ID	: Specifies channel 1.						
Others	: Setting is not allowed.						
Return value	R_A_AS_SUCCESS: Normal termination R_A_AS_ERR_INVALID_ARG: Abnormal termination (The id or p_info member variable of the structure r_a_as_info_t is not specified.) R_A_AS_ERR_ACCESS: Abnormal termination (The channel is already opened.)						
Note	Before calling this function, be sure to configure and activate the multi-protocol encoder interface by using the EC-Lib. This function configures the encoder interface. If this function is executed after power for the encoder has been switched on, send CDF8 eight consecutive times to clear the status flag after executing this function. Calling this API function from within a callback function is not allowed.						

4.5.2 R_A_AS_Close

R_A_AS_Close							
Synopsis	Stops controlling operation of the encoder						
Header	r_a_as_rzt2_if.h						
Declaration	int32_t R_A_AS_Close(const int32_t id);						
Description	This function stops controlling operation of the encoder on the designated channel.						
Argument	id : Designates the ID code to be used. (It is defined in r_a_as_rzt2_dat.h.) <table border="0" style="margin-left: 20px;"> <tr> <td>R_A_AS0_ID</td> <td>: Specifies Channel 0.</td> </tr> <tr> <td>R_A_AS1_ID</td> <td>: Specifies Channel 1.</td> </tr> <tr> <td>Others</td> <td>: Setting is not allowed.</td> </tr> </table>	R_A_AS0_ID	: Specifies Channel 0.	R_A_AS1_ID	: Specifies Channel 1.	Others	: Setting is not allowed.
R_A_AS0_ID	: Specifies Channel 0.						
R_A_AS1_ID	: Specifies Channel 1.						
Others	: Setting is not allowed.						
Return Value	R_A_AS_SUCCESS: Normal termination R_A_AS_ERR_INVALID_ARG: Abnormal termination (The id is not specified.) R_A_AS_ERR_ACCESS: Abnormal termination (A request is being transmitted.)						
Note	In case completion of data transmission and reception (RXEND) interrupts are disabled, operation to control the encoder terminates even if the transmission of a request is in progress. Refer to the part of "R_A_AS_INTE" in "Table 4.7 User-defined constants used by the A-format driver (r_a_as_rzt2_config.h)". Calling this API function from within a callback function is not allowed.						

4.5.3 R_A_AS_GetVersion

R_A_AS_GetVersion

Synopsis	Acquire the version number of the encoder interface driver.
Header	r_a_as_rzt2_if.h
Declaration	uint32_t R_A_AS_GetVersion(const r_a_as_type_t type);
Description	This function acquires the version number of the A-format driver.
Argument	type : Designate R_A_AS_A_FORMAT
Return value	A major part of the version number is stored in the sixteen MSBs and the minor part of the version number is stored in the sixteen LSBs.
Supplement	If the other type is designated, 0xFFFFFFFF will be returned.
Note	Calling this API function from within a callback function is not allowed.

4.5.4 R_A_AS_Control

R_A_AS_Control

Synopsis	Controlling operation of the encoder.
Header	r_a_as_rzt2_if.h
Declaration	int32_t R_A_AS_Control(const int32_t id, const r_a_as_cmd_t cmd, void *const p_buf);
Description	This function controls operations of the encoder by using the cmd argument. See "4.5.5 A-format Control Commands".
Argument	id : Designates the ID code to be used. (It is defined in r_a_as_rzt2_dat.h.) R_A_AS0_ID : Specifies channel 0. R_A_AS1_ID : Specifies channel 1. Others : Setting is not allowed.
	cmd : Command For the list of the commands, see "Table 4.10 Control Commands of the R_A_AS_Control Function".
	p_buf : Arguments corresponding to each cmd.
Return value	R_A_AS_SUCCESS: Normal termination R_A_AS_ERR_INVALID_ARG: Abnormal termination (The id or cmd is not specified.) See "4.5.5, A-format Control Commands" and "Table 4.10 Control Commands of the R_A_AS_Control Function" for other returned values.
Note	Be sure to call R_A_AS_Open before calling this function. Calling this API function from within a callback function is not allowed.

4.5.5 A-format Control Commands

(1) R_A_AS_CMD_SET_PARAM

R_A_AS_CMD_SET_PARAM	
Synopsis	Sets request information
Header	r_a_as_rzt2_if.h
Declaration	int32_t R_A_AS_Control(const int32_t id, const r_a_as_cmd_t cmd, void *const p_buf);
Description	This function sets request information
Argument	id : Designates the ID code to be used. (It is defined in r_a_as_rzt2_dat.h.) R_A_AS0_ID : Specifies channel 0. R_A_AS1_ID : Specifies channel 1. Others : Setting is not allowed. cmd : Designate R_A_AS_CMD_SET_PARAM. p_buf : Request information Designate the pointer to the structure r_a_as_req_t where the request information is stored. See section "4.14.1(2) r_a_as_req_t" for details.
Return value	R_A_AS_SUCCESS: Normal termination R_A_AS_ERR_INVALID_ARG: Abnormal termination (The id is not specified, p_buf is null, or structure r_a_as_req_t member of the p_buf is not specified.) R_A_AS_ERR_ACCESS: Abnormal termination (The channel is not started.)
Note	This control command is utilized only for setting request information. Transmission of commands to the encoder proceeds by using the following control commands. <ul style="list-style-type: none"> • R_A_AS_CMD_TX_TRG • R_A_AS_CMD_TX_ELC Calling this control command from within a callback function is not allowed.

(2) R_A_AS_CMD_ELC_DISABLE

R_A_AS_CMD_ELC_DISABLE	
Synopsis	Disables input of the ELC event triggers
Header	r_a_as_rzt2_if.h
Declaration	int32_t R_A_AS_Control(const int32_t id, const r_a_as_cmd_t cmd, void *const p_buf);
Description	This function disables event input triggers from the ELC.
Arguments	id : Designates the ID code to be used. (It is defined in r_a_as_rzt2_dat.h.) R_A_AS0_ID : Specifies channel 0. R_A_AS1_ID : Specifies channel 1. Others : Setting is not allowed. cmd : Designate R_A_AS_CMD_ELC_DISABLE. p_buf : Not used. (Designate NULL.)
Return value	R_A_AS_SUCCESS: Input of the ELC event trigger is disabled. R_A_AS_ERR_INVALID_ARG: Abnormal termination (The id is not specified.) R_A_AS_ERR_ACCESS: Abnormal termination (The ELC event trigger is not in progress or the channel is not started.)
Note	Calling this control command from within a callback function is not allowed.

(3) R_A_AS_CMD_TX_TRG**R_A_AS_CMD_TX_TRG**

Synopsis	Starts transmission of requests to the encoder
Header	r_a_as_rzt2_if.h
Declaration	int32_t R_A_AS_Control(const int32_t id, const r_a_as_cmd_t cmd, void *const p_buf);
Description	This function starts transmission of requests to the encoder. In normal reception, a callback function which corresponds to the interrupt source being enabled is called every time a transmission of request is made. See section “4.6.1 a_as_txerr_callback” to “4.6.3 a_as_rxend_callback”
Arguments	id : Designates the ID code to be used. (It is defined in r_a_as_rzt2_dat.h.) R_A_AS0_ID : Specifies channel 0. R_A_AS1_ID : Specifies channel 1. Others : Setting is not allowed. cmd : Designate R_A_AS_CMD_TX_TRG. p_buf : Not used. (Designate NULL.)
Return value	R_A_AS_SUCCESS: Normal termination. R_A_AS_ERR_INVALID_ARG: Abnormal termination (The id is not specified.) R_A_AS_ERR_BUSY: Abnormal termination (Transfer is in progress.) R_A_AS_ERR_ACCESS: Abnormal termination (The channel is not started.)
Note	This control command is utilized only for starting transmission of requests to the encoder. Use this command after setting request information by using control command R_A_AS_CMD_SET_PARAM. Calling this control command from within a callback function is not allowed.

(4) R_A_AS_CMD_TX_ELC

R_A_AS_CMD_TX_ELC	
Synopsis	Starts transmission of requests to the encoders in response to the input of an ELC event trigger.
Header	r_a_as_rzt2_if.h
Declaration	int32_t R_A_AS_Control(const int32_t id, const r_a_as_cmd_t cmd, void *const p_buf);
Description	This function enables input of the ELC event trigger, and starts the transmission of requests to the encoder. In normal reception, a callback function which corresponds to the interrupt source being enabled is called every time a transmission of request is made. See section "4.6.1 a_as_txerr_callback" to "4.6.3 a_as_rxend_callback" An error code R_A_AS_ERR_BUSY is returned if this control command is executed while the ELC event trigger is in progress.
Argument	id : Designates the ID code to be used (It is defined in r_a_as_rzt2_dat.h.) R_A_AS0_ID : Specifies channel 0. R_A_AS1_ID : Specifies channel 1. Others : Setting is not allowed. cmd : Designate R_A_AS_CMD_TX_ELC. p_buf : Not used. (Designate NULL.)
Return value	R_A_AS_SUCCESS: Normal termination R_A_AS_ERR_INVALID_ARG: Abnormal termination (The id is not specified.) R_A_AS_ERR_BUSY: Abnormal termination (Transfer or the ELC event trigger is in progress.) R_A_AS_ERR_ACCESS: Abnormal termination (The channel is not started.)
Note	This control command is utilized only for starting transmission of requests to the encoder. Use this command after setting request information by using control command R_A_AS_CMD_SET_PARAM. Calling this control command from within a callback function is not allowed.

(5) R_A_AS_CMD_SETDF

R_A_AS_CMD_SETDF	
Synopsis	Set type of data frame
Header	r_a_as_rzt2_if.h
Declaration	int32_t R_A_AS_Control(const int32_t id, const r_a_as_cmd_t cmd, void *const p_buf);
Description	This function sets data frame type of the designated encoder.
Arguments	id : Designates the ID code to be used. (It is defined in r_a_as_rzt2_dat.h.) R_A_AS0_ID : Specifies channel 0. R_A_AS1_ID : Specifies channel 1. Others : Setting is not allowed. cmd : Designates R_A_AS_CMD_SETDF p_buf : Data frame information Designate the pointer to the structure where the data frame information is stored. See section "4.14.1(3), r_a_as_setdf_t" for details.
Return value	R_A_AS_SUCCESS: Normal termination R_A_AS_ERR_INVALID_ARG: Abnormal termination (The id is not specified.) R_A_AS_ERR_ACCESS: Abnormal termination (Transfer or the ELC event trigger is in progress.)
Note	This control command is for A-format version 3.0 encoders or later. Calling this control command from within a callback function is not allowed.

4.6 Specification of A-format User-defined Functions

4.6.1 a_as_txerr_callback

a_as_txerr_callback	
Synopsis	Callback function that conveys the result of transmission and reception when a transmission error is generated (TXERR) in normal reception.
Header	r_a_as_rzt2_if.h
Declaration	void a_as_txerr_callback (r_a_as_result_t * p_result);
Description	<p>This is a callback function to be registered by the function R_A_AS_Control (R_A_AS_CMD_SET_PARAM). This function conveys the result of transmission and reception in normal reception. This function is called when an interrupt request is generated in response to a transmission error.</p> <p>This function is the context of the interrupt handler. To ensure interrupt responsiveness, return quickly. The function name is an example and can be set freely.</p>
Argument	<p>p_result : Result of transmission and reception</p> <p>This is the pointer to the array where the result of transmission and reception declared by the structure r_a_as_result_t is stored.</p> <p>See “Table 4.3 Result of Transmission and Reception Stored in Each Array Element” for each element.</p> <p>The result of transmission and reception for the encoder address specified by the function R_A_AS_Control(R_A_AS_CMD_SET_PARAM) is updated.</p> <p>See “Table 4.4 Results of Transmission and Reception”.</p>
Return value	None
Note	<p>If the ELC event trigger is enabled, the result of transmission and reception should be acquired within the set timer interval.</p> <p>Depending on the timing of response from the encoder, this callback function may be called even if the function R_A_AS_Close, or R_A_AS_Control (R_A_AS_CMD_ELC_DISABLE) has been executed.</p>

4.6.2 a_as_rxset_callback

a_as_rxset_callback

Synopsis	Callback function that conveys the result of transmission and reception when the setting of received data is complete (RXSET) in normal reception.
Header	r_a_as_rzt2_if.h
Declaration	void a_as_rxset_callback(r_a_as_result_t * p_result);
Description	<p>This is a callback function to be registered by the function R_A_AS_Control(R_A_AS_CMD_SET_PARAM). This function conveys the result of transmission and reception in normal reception. This function is called when an interrupt request is generated in response to completion of setting of the received data.</p> <p>This function is the context of the interrupt handler. To ensure interrupt responsiveness, return quickly. The function name is an example and can be set freely.</p>
Argument	<p>p_result : Result of transmission and reception</p> <p>This is the pointer to the array where the result of transmission and reception declared by the structure r_a_as_result_t.</p> <p>See “Table 4.3 Result of Transmission and Reception Stored in Each Array Element” for the contents of each element. The result of transmission and reception for the encoder address specified by the function R_A_AS_Control(R_A_AS_CMD_SET_PARAM) is updated.</p> <p>Commands for multiple encoder addresses update the results of transmission and reception for encoder category ENC1.</p> <p>See “Table 4.4 Results of Transmission and Reception”.</p>
Return value	None
Note	<p>If the timer operation or input of the ELC event trigger is enabled, the result of transmission and reception should be acquired within the set timer interval.</p> <p>Depending on the timing of response from the encoder, this callback may be called even if the function R_A_AS_Close, or R_A_AS_Control(R_A_AS_CMD_ELC_DISABLE) has been executed.</p>

4.6.3 a_as_rxend_callback

a_as_rxend_callback

Synopsis	Callback function that conveys the result of transmission and reception when the reception of data has been completed (RXEND) in normal reception.
Header	r_a_as_rzt2_if.h
Declaration	void a_as_rxend_callback (r_a_as_result_t * p_result);
Description	<p>This is a callback function to be registered by the function R_A_AS_Control (R_A_AS_CMD_SET_PARAM). This function conveys the result of transmission and reception in normal reception. This function is called when an interrupt request is generated in response to the completion of data reception.</p> <p>This function is the context of the interrupt handler. To ensure interrupt responsiveness, return quickly. The function name is an example and can be set freely.</p>
Argument	<p>p_result : Result of transmission and reception</p> <p>This is the pointer to the array where the result of transmission and reception declared by the structure r_a_as_result_t is stored.</p> <p>See “Table 4.3 Result of Transmission and Reception Stored in Each Array Element” for the contents of each element. The result of transmission and reception for the encoder address specified by the function R_A_AS_Control(R_A_AS_CMD_SET_PARAM) is updated.</p> <p>See “Table 4.4 Results of Transmission and Reception”.</p>
Return value	None
Note	<p>If the timer operation or input of the ELC event trigger is enabled, the result of transmission and reception should be acquired within the set timer interval.</p> <p>Depending on the timing of response from the encoder, this callback may be called even if the function R_A_AS_Close or R_A_AS_Control(R_A_AS_CMD_ELC_DISABLE) has been executed.</p>

4.6.4 a_as_elctimer_callback

a_as_elctimer_callback

Synopsis	Callback function that conveys the result of transmission and reception when the data reception is complete in continuous operation of the ELC event trigger.
Header	r_a_as_rzt2_if.h
Declaration	void a_as_elctimer_callback(r_a_as_result_t * p_result);
Description	<p>This is a callback function to be registered by the function R_A_AS_Control (R_A_AS_CMD_SET_PARAM). This function conveys the result of transmission and reception in normal reception. This function is called every time when an interrupt request is generated in response to completion of the data reception.</p> <p>This function is the context of the interrupt handler. To ensure interrupt responsiveness, return quickly. The function name is an example and can be set freely.</p>
Argument	<p>p_result : Result of transmission and reception</p> <p>This is the pointer to the array where the result of transmission and reception declared by the structure r_a_as_result_t is stored.</p> <p>See “Table 4.3, Result of Transmission and Reception Stored in Each Array Element” for the contents of each element. The result of transmission and reception for the encoder address specified by the function R_A_AS_Control(R_A_AS_CMD_SET_PARAM) is updated.</p> <p>See “Table 4.4, Results of Transmission and Reception” for details.</p>
Return value	None
Note	<p>If the ELC event trigger is enabled, the result of transmission and reception should be acquired within the set timer interval.</p> <p>Depending on the timing of response from the encoder, this callback may be called even if the function R_A_AS_Close or R_A_AS_Control (R_A_AS_CMD_ELC_DISABLE) has been executed.</p>

Table 4.3 Result of Transmission and Reception Stored in Each Array Element

Array Number	Content
p_result[0]	Result of transmission and reception for the encoder section ENC1.
p_result[1]	Result of transmission and reception for the encoder section ENC2.
p_result[2]	Result of transmission and reception for the encoder section ENC3.
p_result[3]	Result of transmission and reception for the encoder section ENC4.
p_result[4]	Result of transmission and reception for the encoder section ENC5.
p_result[5]	Result of transmission and reception for the encoder section ENC6.
p_result[6]	Result of transmission and reception for the encoder section ENC7.
p_result[7]	Result of transmission and reception for the encoder section ENC8.

Table 4.4 Results of Transmission and Reception

Interrupt Source	Results of Transmission and Reception (Member variables of p_result)		
	result	data	status
Occurrence of a transmission error (TXERR)	Only valid when used in the callback functions	Invalid	Only valid when used in the callback functions.
Completion of the setting of received data (RXSET)	Only valid when used in the callback functions.	Valid *1	Only valid when used in the callback functions.
Completion of the transmission and reception of data (RXEND)	Only valid when used in the callback functions.	Valid *1 *2	Only valid when used in the callback functions.

Note: 1. The result of data reception is valid until the transmission of the next request if the ELC event trigger is disabled. Otherwise, the result of data reception is valid until generation of the next ENCIF_INT0, ENCIF_INT4 interrupt.
 2. If TXERR has occurred, it will be invalid.

4.7 A-format Interrupt Handler

4.7.1 a_as0_int_isr

a_as0_int_isr

Synopsis	Interrupt handler for channel 0 data reception completion
Header	-
Declaration	void a_as0_int_isr(void);
Description	Interrupt handler for the following interrupt factors of A_AS 1. data transmission/reception completion interrupt 2. transmission error interrupt (transmission of undefined command) 3. receive data setting completion interrupt
Argument	None
Return value	None

4.7.2 a_as1_int_isr

a_as1_int_isr

Synopsis	Interrupt handler for channel 1 data reception completion
Header	-
Declaration	void a_as1_int_isr(void);
Description	Interrupt handler for the following interrupt factors of A_AS 1. data transmission/reception completion interrupt 2. transmission error interrupt (transmission of undefined command) 3. receive data setting completion interrupt
Argument	None
Return value	None

4.8 Specifications of EnDat API Functions

4.8.1 R_ENDAT_Open

R_ENDAT_Open	
Synopsis	Starting control of the encoder
Header	r_endat_rzt2_if.h
Declaration	r_endat_err_t R_ENDAT_Open(const int32_t id, r_endat_info_t* p_info);
Description	<p>EnDat Driver initializes following initial settings.</p> <ol style="list-style-type: none"> 1. Initial settings of the noise filter 2. Initialization of the encoder (Encoder with a battery unit is not supported) 3. Settings of propagation delay compensation <p>Execute this function 1.3 seconds after the encoder is turned on. The propagation delay of the cable is automatically measured, but if any of the R_ENDAT_CABLE_DELAY measurements fail, the number of measurements is reduced by that amount. If all measurements fail, the value ENDAT_ERR_DRV is returned.</p>
Arguments	<p>id : Specifies the ID to be used. (It is defined in r_endat_rzt2_dat.h.)</p> <p style="margin-left: 20px;">R_ENDAT0_ID : Specifies channel 0.</p> <p style="margin-left: 20px;">R_ENDAT1_ID : Specifies channel 1.</p> <p style="margin-left: 20px;">Others : Setting is not allowed.</p> <p>p_info : Specifies encoder information. Specify the address of the structure r_endat_info_t that contains the encoder information</p>
Return value	<p>ENDAT_SUCCESS : Normal termination</p> <p>ENDAT_ERR_INVALID_ARG : Abnormal termination (The id or r_endat_info_t structure member of the p_info is not specified.)</p> <p>ENDAT_ERR_ACCESS : Abnormal termination (The driver is already open.)</p> <p>ENDAT_ERR_DRV : Abnormal termination (The initialization failed.)</p>
Note	<p>Before executing this function, be sure to configure and start Multi-Protocol Encoder IF using EC-Lib.</p> <p>The encoder initialization process includes sending the mode command "Encoder receive reset", reading Word 13: "Number of clocks", clearing Word 0 "Error messages", and clearing Word 1: "Warning messages". When initializing the encoder with a battery unit, add the required procedures to the encoder after executing this function by referring to the "Power-on procedure" in the HEIDENHAIN application note (v03).</p>

4.8.2 R_ENDAT_Close

R_ENDAT_Close	
Synopsis	Ending control of the EnDat encoder
Header	r_endat_rzt2_if.h
Declaration	r_endat_err_t R_ENDAT_Close(const int32_t id);
Description	This function handles ending of the encoder IF driver.
Arguments	<p>id : Specifies the ID to be used. (It is defined in r_endat_rzt2_dat.h.)</p> <p style="margin-left: 20px;">R_ENDAT0_ID : Specifies channel 0</p> <p style="margin-left: 20px;">R_ENDAT1_ID : Specifies channel 1</p> <p style="margin-left: 20px;">Other than above : Setting is not allowed</p>
Return value	<p>ENDAT_SUCCESS : Normal termination</p> <p>ENDAT_ERR_INVALID_ARG : Abnormal termination (The id is not specified.)</p> <p>ENDAT_ERR_ACCESS : Abnormal termination (Transfer is in progress.)</p>

4.8.3 R_ENDAT_GetVersion

R_ENDAT_GetVersion

Synopsis	Acquiring the version number of the encoder interface driver	
Header	r_endat_rzt2_if.h	
Declaration	uint32_t R_ENDAT_GetVersion(void);	
Description	This function acquires the version number of the EnDat driver.	
Arguments	None	
Return value	Version information	: The major and minor parts of the version number are stored in the sixteen MSBs and sixteen LSBs, respectively. Ex.) For ver.1.2, the value returned is 0x00010002

4.8.4 R_ENDAT_Control

R_ENDAT_Control

Synopsis	Controlling the EnDat encoder	
Header	r_endat_rzt2_if.h	
Declaration	r_endat_err_t R_ENDAT_Control(const int32_t id, const r_endat_cmd_t cmd, void *const p_buf);	
Description	Use the argument cmd to control the EnDat encoder. See "4.8.5 EnDat Control Commands" for the operation of the control commands.	
Arguments	id	: Specifies the ID to be used. (It is defined in r_endat_rzt2_dat.h.) R_ENDAT0_ID : Specifies channel 0. R_ENDAT1_ID : Specifies channel 1. Others : Setting is not allowed.
	cmd	: Command For details, see section "4.14.6(2) r_endat_cmd_t".
	p_buf	: Arguments for each cmd
Return value	ENDAT_SUCCESS	: Normal termination
	ENDAT_ERR_INVALID_ARG	: Abnormal termination (The id or cmd is not specified.) For other return values, see "4.8.5 EnDat Control Commands".
Note	Be sure to execute R_ENDAT_Open before executing this function.	

4.8.5 EnDat Control Commands

(1) ENDAT_CMD_REQ

ENDAT_CMD_REQ	
Synopsis	Sends requests to the EnDat encoder
Header	r_endat_rzt2_if.h
Declaration	r_endat_err_t R_ENDAT_Control(const int32_t id, const r_endat_cmd_t cmd, void *const p_buf);
Description	<p>Sends requests to the EnDat encoder.</p> <p>The endat_callback function is called once for each request sent.</p> <p>If the Continuous mode setting is enabled, the endat_poscon_callback function is repeatedly called until ENDAT_CMD_POS_STOP is executed and the endat_rdst_callback function is called.</p> <p>If the ELC mode setting is enabled, the request is repeatedly sent in synchronize with ELC events until ENDAT_CMD_POS_STOP is executed and the endat_rdst_callback function is called. Each time a request is sent, the endat_poscon_callback function is called.</p>
Arguments	<p>id : Specifies the ID to be used. (It is defined in r_endat_rzt2_dat.h.)</p> <p style="padding-left: 20px;">R_ENDAT0_ID : Specifies channel 0.</p> <p style="padding-left: 20px;">R_ENDAT1_ID : Specifies channel 1.</p> <p style="padding-left: 20px;">Others : Setting is not allowed.</p> <p>cmd : Designate ENDAT_CMD_REQ.</p> <p>p_buf : Request information</p> <p style="padding-left: 20px;">Specifies the pointer to the r_endat_req_t structure which holds the request information. For details of the r_endat_req_t structure, see section "4.14.4(3) r_endat_req_t".</p>
Return value	<p>R_ENDAT_SUCCESS : Normal termination</p> <p>R_ENDAT_ERR_INVALID_ARG : Abnormal termination (The id or cmd is not specified, p_buf is NULL, or the structure r_endat_req_t member is not specified.)</p> <p>R_ENDAT_ERR_BUSY : Abnormal termination (Transfer is in progress.)</p> <p>R_ENDAT_ERR_ACCESS : Abnormal termination (The channel is not started.)</p>

(2) ENDAT_CMD_POS_STOP**ENDAT_CMD_POS_STOP**

Synopsis	Stop continuous acquisition of position values
Header	r_endat_rzt2_if.h
Declaration	r_endat_err_t R_ENDAT_Control(const int32_t id, const r_endat_cmd_t cmd, void *const p_buf);
Description	Disables the Continuous mode setting during reception processing in Continuous mode, and disables the ELC mode setting during event-synchronized send/receive processing in ELC mode, thus continuous reception of position values from the EnDat encoder is stopped. An error is returned if there is no continuous reception processing of position values.
Arguments	id : Specifies the ID to be used. (It is defined in r_endat_rzt2_dat.h.) R_ENDAT0_ID : Specifies channel 0. R_ENDAT1_ID : Specifies channel 1. Others : Setting is not allowed. cmd : Designate ENDAT_CMD_POS_STOP. p_buf : Not used. (Designate NULL.)
Return value	R_ENDAT_SUCCESS : Normal termination R_ENDAT_ERR_INVALID_ARG : Abnormal termination (The id or cmd is not specified.) R_ENDAT_ERR_ACCESS : Abnormal termination (Continuous mode or ELC mode request is not sent.)

4.9 Specifications of EnDat User-defined Functions**4.9.1 enc_init_reset_wait_callback****enc_init_reset_wait_callback**

Synopsis	Function to generate wait time after encoder reset
Header	r_endat_rzt2_if.h
Declaration	void enc_init_reset_wait_callback(void);
Description	Callback function to be registered with the R_ENDAT_Open function. Initialization process of the connected encoder generates the time to wait after the encoder reset process. Set 60 ms waiting time or longer. The function name is an example and can be freely set.
Arguments	None
Return value	None

4.9.2 enc_init_mem_wait_callback

enc_init_mem_wait_callback

Synopsis	Function to generate wait time for detecting memory area selection timeout
Header	r_endat_rzt2_if.h
Declaration	void enc_init_mem_wait_callback(void);
Description	<p>Callback function to be registered with the R_ENDAT_Open function. Generates a wait time used for detecting a timeout error in the process of selecting a memory area in the initialization process of the connected encoder. Set 743 us* waiting time or longer. The function name is an example and can be freely set.</p> <p>Note: This value is based on the assumption of (2 clock cycles + mode command(6 clock cycles) + MRS code(8 clock cycles) + 16 clock cycles + 2T(2 clock cycles) + maximum 7 clock cycles + Start(1 clock cycle) + MRS code(8 clock cycles) + 16 clock cycles + CRC(5 clock cycles))\times(1/100 kHz) + t_m(30 us) + t_R(0.5 us) + t_D(1.7 us) = 742.2 us.</p> <p>The transmission clock frequency is set to 100kHz in the driver during the encoder initialization process. The delay time t_D assumes a cable length of 150 m. Users are required to adjust the waiting time according to the encoder and the cable length.</p>
Arguments	None
Return value	None

4.9.3 enc_init_pram_wait_callback

enc_init_pram_wait_callback

Synopsis	Function to generate wait time for detecting parameter transmission timeout
Header	r_endat_rzt2_if.h
Declaration	void enc_init_pram_wait_callback(void);
Description	<p>Callback function to be registered with the R_ENDAT_Open function. Generates a wait time for the initialization process of the connected encoder to detect timeout errors in the process of sending and receiving parameters by the encoder. Set 13 ms* waiting time or longer. The function name is an example and can be freely set.</p> <p>Note: Assumes memory access time (12 ms) + (Start(1 clock cycle) + Address(8 clock cycles) + Parameters(16 clock cycles) + CRC(5 clock cycles))\times(1/100 kHz) + t_m(30 us) + t_R(0.5 us) + t_D(1.7 us) = 12.33 ms.</p> <p>During the encoder initialization process, the transmission clock frequency is set to 100 kHz in the driver. The delay time t_D assumes a cable length of 150 m. Users are required to adjust the waiting time according to the encoder and the cable length.</p>
Arguments	None
Return value	None

4.9.4 enc_init_cable_wait_callback

enc_init_cable_wait_callback

Synopsis	Function to generate wait time for detecting propagation delay measurement timeout
Header	r_endat_rzt2_if.h
Declaration	void enc_init_cable_wait_callback(void);
Description	<p>Callback function to be registered with the R_ENDAT_Open function. Generates a wait time used for detecting a timeout error in the process of measuring cable propagation delay in the initialization process of the connected encoder. Set 588 us* waiting time or longer. The function name is an example and can be freely set.</p> <p>Note: Assumes $t_{cal}(5 \text{ us}) + (\text{Start}(1 \text{ clock cycle}) + \text{Error}(1 \text{ clock cycle}) + \text{maximum number of bits of main received data (48 bits) + number of CRC bits (5 bits)}) \times (1/100 \text{ kHz}) + t_m(30 \text{ us}) + t_R(0.5 \text{ us}) + t_D(1.7 \text{ us}) = 587.2 \text{ us}$.</p> <p>During the encoder initialization process, the transmission clock frequency is set to 100 kHz in the driver. The delay time t_D assumes a cable length of 150 m. Users are required to adjust the waiting time according to the encoder and the cable length.</p>
Arguments	None
Return value	None

4.9.5 endat_callback

endat_callback

Synopsis	Data reception result notification function for sending requests
Header	r_endat_rzt2_if.h
Declaration	void endat_callback(r_endat_result_t * p_result, r_endat_protocol_err_t *p_err);
Description	<p>This callback function is registered with the R_ENDAT_Control(ENDAT_CMD_REQ) function. This function is called when the MBERR interrupt, the WDG interrupt or the RXEND interrupt is occurred.</p> <p>This function is the context of the interrupt handler. To ensure interrupt responsiveness, return immediately. The function name is an example and can be freely set.</p>
Argument	<p>p_result : Result of sending/receiving Pointer to structure r_endat_result_t that stores the result of sending/receiving data. The data reception result is valid until the next request is sent.</p> <p>p_err : Error information Pointer to the structure r_endat_protocol_err_t that contains the results of sending and receiving. The data reception result is valid until the next request is sent.</p>
Return value	None

4.9.6 endat_poscon_callback

endat_poscon_callback	
Synopsis	Data reception result notification function for sending requests (Continuous mode, ELC mode)
Header	r_endat_rzt2_if.h
Declaration	void endat_poscon_callback(r_endat_result_t * p_result, endat_protocol_err_t *p_err);
Description	This callback function is registered with the R_ENDAT_Control (ENDAT_CMD_REQ) function when data transmission is performed in Continuous mode or ELC mode. This function notifies the result of data reception in response to a request, and is called when the MBERR interrupt, the WDG interrupt or the RXEND interrupt is occurred. This function is the context of the interrupt handler. To ensure interrupt responsiveness, return immediately. The function name is an example and can be freely set.
Arguments	<p>p_result : Result of sending/receiving Pointer to the structure r_endat_result_t that stores the result of sending/receiving data. The data reception result is valid until the next request is sent/received.</p> <p>p_err : Error information Pointer to the structure r_endat_protocol_err_t that stores the results of sending and receiving. The data reception result is valid until the next request is sent/received.</p>
Return value	None

4.9.7 endat_rdst_callback

endat_rdst_callback	
Synopsis	Callback function to notify that the next data communication is ready to start
Header	r_endat_rzt2_if.h
Declaration	void endat_rdst_callback(void);
Description	This callback function is registered with the R_ENDAT_Control(ENDAT_CMD_REQ) function. It is called after the endat_callback function when an RDSTC interrupt occurs. While operating in Continuous mode or ELC mode, this function is called after the endat_poscon_callback function each time data reception is completed. This function is the context for the interrupt handler. To ensure interrupt responsiveness, return immediately. The function name is an example and can be freely set.
Arguments	None
Return value	None

4.10 EnDat Interrupt Handler

4.10.1 endat0_rx_int_isr

endat0_rx_int_isr	
Synopsis	Channel 0 Data reception completion interrupt handler
Header	-
Declaration	void endat0_rx_int_isr(void);
Description	Interrupt handler for the following interrupt factors on EnDat channel 0. 1. MBERR interrupt 2. WDG interrupt 3. RXEND interrupt 4. RDSTC interrupt
Arguments	None
Return value	None

4.10.2 endat1_rx_int_isr

endat1_rx_int_isr	
Synopsis	Channel 1 Data reception completion interrupt handler
Header	-
Declaration	void endat1_rx_int_isr(void);
Description	Interrupt handler for the following interrupt factors on EnDat channel 1. <ol style="list-style-type: none"> 1. MBERR interrupt 2. WDG interrupt 3. RXEND interrupt 4. RDSTC interrupt
Arguments	None
Return value	None

4.11 Interrupts

Table 4.5 lists the interrupt for the Dual Encoder driver. If the A-format channel and the EnDat channel are switched, EnDat interrupt is assigned in ENCIF_INT0 and A-format interrupt is assigned in ENCIF_INT4.

Table 4.5 Interrupt for the Dual Encoder Driver

Interrupt	ID	Outline
ENCIF_INT0	372	Interrupts are generated by the following interrupt factors of the Ch0 A-format encoder. <ol style="list-style-type: none"> 1. data transmission and reception completion interrupt 2. Transmission error interrupt (transmission of undefined command) 3. Received data setting completion interrupt
ENCIF_INT4	376	Interrupts are generated by the following interrupt factors of the Ch1 EnDat encoder. <ol style="list-style-type: none"> 1. MBERR interrupt 2. WDG interrupt 3. RXEND interrupt 4. RDSTC interrupt

4.12 Constant and Error Code

Table 4.6 lists the constant/error code definition tables. For each definition, refer to the respective table. For EnDat error codes, see "4.14.6(1) r_endat_err_t".

Table 4.6 Constant/error code definition tables

Table number	Contents
Table 4.7	User-defined constants used by the A-format driver (r_a_as_rzt2_config.h)
Table 4.8	A-format Driver Type
Table 4.9	Methods of Connection between A_AS and Encoders
Table 4.10	Control Commands of the R_A_AS_Control Function
Table 4.11	A-format Bitrate
Table 4.12	A-format Encoder Addresses
Table 4.13	A-format Commands
Table 4.14	Command Data Frame Selection
Table 4.15	A-format Error Codes
Table 4.16	User-defined Constants to be Used in the EnDat Driver (r_endat_rzt2_config.h)
Table 4.17	EnDat 2.2 Mode Commands
Table 4.18	EnDat Transmission Clock Frequencies
Table 4.19	EnDat Watchdog Timer Time Units
Table 4.20	EnDat Low-level Period at the Start of Data Transmission
Table 4.21	EnDat MRS Codes

Table 4.7 User-defined constants used by the A-format driver (r_a_as_rzt2_config.h)

Constant Name	Setting	Description
R_A_AS_INTE	0xC0	The value of the interrupt enable register (setting this register enables interrupt sources RXEND and TXERR)
R_A_AS_NFINTV_2500KBPS	0x00	The NFINTV value when the bit rate is 2.5 Mbps. *
R_A_AS_NFINTV_4MBPS	0x00	The NFINTV value when the bit rate is 4 Mbps. *
R_A_AS_NFINTV_6670KBPS	0x00	The NFINTV value when the bit rate is 6.67 Mbps. *
R_A_AS_NFINTV_8MBPS	0x00	The NFINTV value when the bit rate is 8 Mbps. *
R_A_AS_NFSCNT_2500KBPS	0x07	The NFSCNT value when the bit rate is 2.5 Mbps. *
R_A_AS_NFSCNT_4MBPS	0x04	The NFSCNT value when the bit rate is 4 Mbps. *
R_A_AS_NFSCNT_6670KBPS	0x02	The NFSCNT value when the bit rate is 6.67 Mbps. *
R_A_AS_NFSCNT_8MBPS	0x01	The NFSCNT value when the bit rate is 8 Mbps. *
R_A_AS_T2_ONE_2500KBPS	0x0014	The value set in the T2 register when the connection is one-to-one, and the bit rate is 2.5 Mbps. *
R_A_AS_T2_ONE_4MBPS	0x0014	The value set in the T2 register when the connection is one-to-one, and the bit rate is 4 Mbps. *
R_A_AS_T2_ONE_6670KBPS	0x0014	The value set in the T2 register when the connection is one-to-one, and the bit rate is 6.67 Mbps. *
R_A_AS_T2_ONE_8MBPS	0x0014	The value set in the T2 register when the connection is one-to-one, and the bit rate is 8 Mbps. *
R_A_AS_T2_BUS_2500KBPS	0x0096	The value set in the T2 register when the connection is a bus connection, and the bit rate is 2.5 Mbps. *
R_A_AS_T2_BUS_4MBPS	0x0064	The value set in the T2 register when the connection is a bus connection, and the bit rate is 4 Mbps. *
R_A_AS_T2_BUS_6670KBPS	0x0040	The value set in the T2 register when the connection is a bus connection, and the bit rate is 6.67 Mbps. *
R_A_AS_T2_BUS_8MBPS	0x003C	The value set in the T2 register when the connection is a bus connection, and the bit rate is 8 Mbps. *
R_A_AS_T3_2500KBPS	0x001E	The value set in the T3 register when the bit rate is 2.5 Mbps. *
R_A_AS_T3_4MBPS	0x0014	The value set in the T3 register when the bit rate is 4 Mbps. *
R_A_AS_T3_6670KBPS	0x0014	The value set in the T3 register when the bit rate is 6.67 Mbps.*
R_A_AS_T3_8MBPS	0x000E	The value set in the T3 register when the bit rate is 8 Mbps. *
R_A_AS_T4	0x00C8	The value set in the T4 register
R_A_AS_T5_2500KBPS	0x003C	The value set in the T5 register when the bit rate is 2.5 Mbps. *
R_A_AS_T5_4MBPS	0x0028	The value set in the T5 register when the bit rate is 4 Mbps. *
R_A_AS_T5_6670KBPS	0x0028	The value set in the T5 register when the bit rate is 6.67 Mbps.*
R_A_AS_T5_8MBPS	0x001E	The value set in the T5 register when the bit rate is 8 Mbps. *
R_A_AS_T9_ONE	0x005E	The value set in the T9 register when the connection is one-to-one.
R_A_AS_T9_BUS	0x00C2	The value set in the T9 register when the connection is a bus connection.

Note. In this sample program, the values set in each register are the recommended values.

Table 4.8 A-format Driver Type

Constant Name	Setting	Description
R_A_AS_A_FORMAT	0	Designates the A-format driver.

Table 4.9 Methods of Connection between A_AS and Encoders

Constant Name	Setting	Description
R_A_AS_ONE_FOR_ONE	0	One-to-one connection
R_A_AS_BUS	1	Bus connection

Table 4.10 Control Commands of the R_A_AS_Control Function

Constant Name	Setting	Description
R_A_AS_CMD_SET_PARAM	0xAF000000	Sets requests information
R_A_AS_CMD_ELC_DISABLE	0xAF000002	Disable input of the ELC event trigger
R_A_AS_CMD_TX_TRG	0xAF000003	Starts transmission of a command by timer operation.
R_A_AS_CMD_TX_ELC	0xAF000005	Starts transmission of a command by input of the ELC event trigger
R_A_AS_CMD_SETDF	0xAF000006	Sets data frame types for A-format version 3.0 encoders or later.

Table 4.11 A-format Bitrate

Constant Name	Setting	Description
R_A_AS_2500KBPS	0	2.5 Mbps
R_A_AS_4MBPS	1	4 Mbps
R_A_AS_6670KBPS	2	6.67 Mbps
R_A_AS_8MBPS	3	8 Mbps

Table 4.12 A-format Encoder Addresses

Constant Name	Setting	Description
R_A_AS_ECN1	0	The address of the encoder section ENC1.
R_A_AS_ECN2	1	The address of the encoder section ENC2.
R_A_AS_ECN3	2	The address of the encoder section ENC3.
R_A_AS_ECN4	3	The address of the encoder section ENC4.
R_A_AS_ECN5	4	The address of the encoder section ENC5.
R_A_AS_ECN6	5	The address of the encoder section ENC6.
R_A_AS_ECN7	6	The address of the encoder section ENC7.
R_A_AS_ECN8	7	The address of the encoder section ENC8.

Table 4.13 A-format Commands

Constant Name	Setting	Description
R_A_AS_CDF0	0	Defined for the command data frame CDF0.
R_A_AS_CDF1	1	Defined for the command data frame CDF1.
R_A_AS_CDF2	2	Defined for the command data frame CDF2.
R_A_AS_CDF3	3	Defined for the command data frame CDF3.
R_A_AS_CDF4	4	Defined for the command data frame CDF4.
R_A_AS_CDF5	5	Defined for the command data frame CDF5.
R_A_AS_CDF6	6	Defined for the command data frame CDF6.
R_A_AS_CDF7	7	Defined for the command data frame CDF7.
R_A_AS_CDF8	8	Defined for the command data frame CDF8.
R_A_AS_CDF9	9	Defined for the command data frame CDF9.
R_A_AS_CDF10	10	Defined for the command data frame CDF10.
R_A_AS_CDF11	11	Defined for the command data frame CDF11.
R_A_AS_CDF12	12	Defined for the command data frame CDF12.
R_A_AS_CDF13	13	Defined for the command data frame CDF13.
R_A_AS_CDF14	14	Defined for the command data frame CDF14.
R_A_AS_CDF15	15	Defined for the command data frame CDF15.
R_A_AS_CDF16	16	Defined for the command data frame CDF16.
R_A_AS_CDF17	17	Defined for the command data frame CDF17.
R_A_AS_CDF18	18	Defined for the command data frame CDF18.
R_A_AS_CDF19	19	Defined for the command data frame CDF19.
R_A_AS_CDF20	20	Defined for the command data frame CDF20.
R_A_AS_CDF21	21	Defined for the command data frame CDF21.
R_A_AS_CDF22	22	Defined for the command data frame CDF22.
R_A_AS_CDF23	23	Defined for the command data frame CDF23. *1
R_A_AS_CDF24	24	Defined for the command data frame CDF24. *1
R_A_AS_CDF25	25	Defined for the command data frame CDF25. *1
R_A_AS_CDF26	26	Defined for the command data frame CDF26. *1
R_A_AS_CDF27	27	Defined for the command data frame CDF27.
R_A_AS_CDF28	28	Defined for the command data frame CDF28.
R_A_AS_CDF29	29	Defined for the command data frame CDF29.
R_A_AS_CDF30	30	Defined for the command data frame CDF30.
R_A_AS_CDF13x	113	Defined for the command data frame CDF13 (FC=11). *1
R_A_AS_CDF14x	114	Defined for the command data frame CDF14 (FC=11). *1
R_A_AS_CDF16x	116	Defined for the command data frame CDF16 (FC=11). *1
R_A_AS_CDF18x	118	Defined for the command data frame CDF18 (FC=11). *1
R_A_AS_CDFx_OFFSET	100	Defined for offset of FC=11 commands against FC=00.

Note: 1. It is command data frame for A-format version 3.0 encoders or later.

Table 4.14 Command Data Frame Selection

Constant Name	Setting	Description *1
R_A_AS_SET_CDF1	0	Type of the command data frames CDF1 and CDF5.
R_A_AS_SET_CDF8	1	Type of the command data frames CDF8 to CDF12.

Note: 1. Setting of the data frame types is available for A-format version 3.0 encoders or later.

Table 4.15 A-format Error Codes

Constant Name	Setting	Description
R_A_AS_SUCCESS	0	Normal termination
R_A_AS_ERR_INVALID_ARG	-1	Argument error
R_A_AS_ERR_BUSY	-2	The API cannot be executed.
R_A_AS_ERR_ACCESS	-3	API execution order error.

Table 4.16 User-defined Constants to be Used in the EnDat Driver (r_endat_rzt2_config.h)

Constant	Set value	Content
R_ENDAT_CABLE_DELAY	5	The number of times the propagation delay is automatically measured. Set it to 5 to 255 times.
R_ENDAT_ADD_NUM	0u	Number of additional information to receive

Table 4.17 EnDat 2.2 Mode Commands

Constant	Value	Content
R_ENDAT_POS	0x07u	"Encoder send position values" command
R_ENDAT_MEM	0x0Eu	"Selection of memory area" command
R_ENDAT_RX_PARAM	0x1Cu	"Encoder receive parameter" command
R_ENDAT_PARAM	0x23u	"Encoder send parameter" command
R_ENDAT_RESET	0x2Au	"Encoder receive reset" command
R_ENDAT_POS_ADD_DATA	0x38u	"Encoder send position values with additional data" command
R_ENDAT_POS_MEM	0x09u	"Encoder send position values and selection of the memory area" command
R_ENDAT_POS_RX_PARAM	0x1Bu	"Encoder send position values and receive parameter" command
R_ENDAT_POS_PARAM	0x24u	"Encoder send position values and send parameter" command
R_ENDAT_POS_RX_ERR_RESET	0x2Du	"Encoder send position values and receiver error reset" command

Note: For details, refer to the "EnDat Specification" which is available from HEIDENHAIN on request.

Table 4.18 EnDat Transmission Clock Frequencies

Constant	value	content
R_ENDAT_FTCLK_16670	0x3u	16.67 MHz *
R_ENDAT_FTCLK_8330	0x6u	8.33 MHz *
R_ENDAT_FTCLK_4160	0xBu	4.16 MHz *
R_ENDAT_FTCLK_4000	0x8u	4 MHz *
R_ENDAT_FTCLK_2000	0xCu	2 MHz
R_ENDAT_FTCLK_1000	0xDu	1 MHz
R_ENDAT_FTCLK_200	0xEu	0.2 MHz
R_ENDAT_FTCLK_100	0xFu	0.1 MHz

Note: Propagation delay compensation should be enabled (delay_comp=true).

Table 4.19 EnDat Watchdog Timer Time Units

Constant	value	content
R_ENDAT_WD_RANGE_US	0x00u	Watchdog Timer time unit is microseconds
R_ENDAT_WD_RANGE_MS	0x80u	Watchdog Timer time unit is milliseconds

Table 4.20 EnDat Low-level Period at the Start of Data Transmission

Constant	value	content
R_ENDAT_TST_HALF_TCLK	0x00u	1/2 TCLK
R_ENDAT_TST_500NS	0x01u	0.5 us *
R_ENDAT_TST_1US	0x02u	1 us *
R_ENDAT_TST_1500NS	0x03u	1.5 us *
R_ENDAT_TST_2US	0x04u	2 us *
R_ENDAT_TST_4US	0x05u	4 us *
R_ENDAT_TST_8US	0x06u	8 us *
R_ENDAT_TST_10US	0x07u	10 us *

Note The low-level period has a margin of error. See the hardware manual for details.

Table 4.21 EnDat MRS Codes

Constant	value	content
R_ENDAT_MRS_INFO1_NOP	0x40u	Send additional info 1 without data contents (NOP)
R_ENDAT_MRS_DIA	0x41u	Send diagnostic values
R_ENDAT_MRS_POS2_LSB	0x42u	Send position value 2, word 1 LSB
R_ENDAT_MRS_POS2_CENTER	0x43u	Send position value 2, word 2
R_ENDAT_MRS_POS2_MSB	0x44u	Send position value 2, word 3 MSB
R_ENDAT_MRS_MEM_LSB	0x45u	Acknowledge memory content LSB
R_ENDAT_MRS_MEM_MSB	0x46u	Acknowledge memory content MSB
R_ENDAT_MRS_MRS_CODE	0x47u	Acknowledge MRS code
R_ENDAT_MRS_TEST_SMD	0x48u	Acknowledge test command
R_ENDAT_MRS_TEST_LSB	0x49u	Send test values, word 1 LSB
R_ENDAT_MRS_TEST_CENTER	0x4Au	Send test values, word 2
R_ENDAT_MRS_TEST_MSB	0x4Bu	Send test values, word 3 MSB
R_ENDAT_MRS_TEMP1	0x4Cu	Send temperature 1
R_ENDAT_MRS_TEMP2	0x4Du	Send temperature 2
R_ENDAT_MRS_ADD_SEN	0x4Eu	Additional sensors
R_ENDAT_MRS_NOT_INFO1	0x4Fu	Stop sending additional datum 1
R_ENDAT_MRS_INFO2_NOP	0x50u	Send additional datum 2 without data contents
R_ENDAT_MRS_COM	0x51u	Send commutation
R_ENDAT_MRS_ACC	0x52u	Send acceleration
R_ENDAT_MRS_COM_ACC	0x53u	Send commutation & acceleration
R_ENDAT_MRS_LIM_POS	0x54u	Send limit position signals
R_ENDAT_MRS_LIM_POS_ACC	0x55u	Send limit position signals & acceleration
R_ENDAT_MRS_ASY_POS_LSB	0x56u	Asynchronous position value, word 1 LSB
R_ENDAT_MRS_ASY_POS_CENTER	0x57u	Asynchronous position value, word 2
R_ENDAT_MRS_ASY_POS_MSB	0x58u	Asynchronous position value, word 3 MSB
R_ENDAT_MRS_OPE_STA_ERR	0x59u	Operating status error sources
R_ENDAT_MRS_TIM_STA	0x5Bu	Timestamp
R_ENDAT_MRS_NOT_INFO2	0x5Fu	Stop sending additional datum 2
R_ENDAT_MRS_OPE_STAT	0xB9u	Operating status
R_ENDAT_MRS_ENC_MANU1	0xA1u	Parameters of the encoder manufacturer 1
R_ENDAT_MRS_ENC_MANU2	0xA3u	Parameters of the encoder manufacturer 2
R_ENDAT_MRS_ENC_MANU3	0xA5u	Parameters of the encoder manufacturer 3
R_ENDAT_MRS_OPE_PARAM	0xA7u	Operating parameters
R_ENDAT_MRS_OEM1	0xA9u	Parameters of the OEM 1
R_ENDAT_MRS_OEM2	0xABu	Parameters of the OEM 2
R_ENDAT_MRS_OEM3	0xADu	Parameters of the OEM 3
R_ENDAT_MRS_OEM4	0xAFu	Parameters of the OEM 4
R_ENDAT_MRS_COMP_VAL1	0xB1u	Compensation Values of the encoder manufacturer 1
R_ENDAT_MRS_COMP_VAL2	0xB3u	Compensation Values of the encoder manufacturer 2
R_ENDAT_MRS_COMP_VAL3	0xB5u	Compensation Values of the encoder manufacturer 3
R_ENDAT_MRS_COMP_VAL4	0xB7u	Compensation Values of the encoder manufacturer 4
R_ENDAT_MRS_PARAM_ENDAT22	0xBDu	Parameters of the encoder manufacturer for EnDat 2.2
R_ENDAT_MRS_PARAM_SEC2	0xBFu	Parameters of the section 2 memory area
R_ENDAT_MRS_OPE_PARAM2	0xBBu	Operating parameters 2

Note: For details, refer to the "EnDat Specification" which is available from HEIDENHAIN on request.

4.13 Fixed-width Integer Types

Table 4.21 lists the fixed-width integers for the sample code. These fixed-width integers are defined in the standard libraries.

Table 4.22 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)

4.14 Structures, Unions, and Enumerated Types

4.14.1 A-format Structures

(1) r_a_as_info_t

Initialization information of the A_AS control unit,

```
typedef struct
{
    uint8_t    connect;    Connection method
                    Designate the method of connection between A_AS and the
                    encoders. See "Table 4.9 Methods of Connection between A_AS
                    and Encoders" for the values to be designated.
                    Note: This setting is reflected in the T2, T3, T5, and T9 registers.

    uint8_t    bitrate;    Bit rate
                    Designate the bit rate for communications with the encoder. See
                    "Table 4.11 A-format Bitrate" for the values to be designated.
                    Note: This setting is reflected in the T2, T3, T5, T9, and BR
                    registers.

    uint16_t   ifmg;       Margin value
                    Designate the timer margin for use in monitoring by the watchdog
                    timer of the time to start receiving the encoder data (IF).
                    Note: This setting is reflected in the IFMG register.
} r_a_as_info_t
```

(2) r_a_as_req_t

Information of requests to be sent to the encoders.

```
typedef struct
{
    uint8_t      encadr;      Encoder address
                        Designate the encoder address. See "Table 4.12 A-format
                        Encoder Addresses" for the value to be designated.
                        This setting is reflected in the EA bit of the TXC register.

    uint8_t      cmd;        Command
                        Designate the command code to be sent to the encoder. See
                        "Table 4.13 A-format Commands" for the value to be
                        designated.
                        If a value other than those listed in the table is designated, a
                        transmission error interrupt is generated upon sending of the
                        value. Additionally, if the value is not listed in the said table and
                        exceeds 0x20, an error with the error code
                        R_A_AS_ERR_INVALID_ARG is generated.
                        Some commands are not available depending on the
                        connection method.

    uint8_t      memadr;     Memory address
                        Designate the address of the memory in the encoder. *1
                        Set this value only in the following cases:
                        cmd = R_A_AS_CDF13
                        cmd = R_A_AS_CDF14
                        cmd = R_A_AS_CDF13x
                        cmd = R_A_AS_CDF14x

    uint8_t      membank;    Memory bank
                        Designate the memory bank of the encoder.
                        Set this value only in the following cases:
                        cmd = R_A_AS_CDF13x
                        cmd = R_A_AS_CDF14x

    uint16_t     memdat;     Data to be written to the memory
                        Designate the data to be written to the memory.
                        Set this value only in the following case:
                        cmd = R_A_AS_CDF14
                        cmd = R_A_AS_CDF14x

    uint32_t     encid;      Recognition code or velocity coefficient
                        Designate the 24-bit recognition code.
                        Set this value only in the following cases:
                        cmd = R_A_AS_CDF18
                        cmd = R_A_AS_CDF19
                        cmd = R_A_AS_CDF20
                        Designate the 19-bit velocity coefficient.
                        Set this value only in the following case:
                        cmd = R_A_AS_CDF18x

    r_a_as_result_cb_t cbadr_txerr The pointer to the callback function to be called in response to
                                an ENCIF_INT interrupt request issued due to TXERR. See
                                "4.6.1 a_as_txerr_callback" for details. *1

    r_a_as_result_cb_t cbadr_rxset; The pointer to the callback function to be called in response to
                                an ENCIF_INT interrupt request issued due to RXSET. See
                                "4.6.2 a_as_rxset_callback" for details. *1

    r_a_as_result_cb_t cbadr_rxend; The pointer to the callback function to be called in response to
                                an ENCIF_INT interrupt request issued due to RXEND. See
                                "4.6.3 a_as_rxend_callback" for details. *1
}
```

```

    bool            pre;           Set true in this variable to set request information while
                                transmission and reception of data by the ELC event trigger is
                                in progress. Set false to set request information in other cases.
                                (true: Set request information while transmission and reception
                                of data by the ELC event trigger is in progress)

} r_a_as_req_t

```

- Note
1. For the command R_A_AS_CDF13, R_A_AS_CDF13x or R_A_AS_CDF14x, the accessible address range is between 0x00 and 0xFF. For the command R_A_AS_CDF14, the accessible address range is between 0x00 and 0xEF.
 2. Callback function will not be run if the pointer is NULL.

(3) r_a_as_setdf_t

Information of the data frame type settings

```

typedef struct
{
    uint8_t         encadr;       Encoder address
                                Designate the encoder address. See "Table 4.12, A-format
                                Encoder Addresses" for the value to be designated.

    bool            encmod;       Data frame type extension for the CDF1 and CDF5
                                false: Data frame for the CDF1 and CDF5 is received as ABS
                                LSB 24-bit.
                                true: Data frame for the CDF1 and CDF5 is received as ABS
                                full 40-bit and velocity.

} r_a_as_result_t

```

(4) r_a_as_result_t

Result of transmission and reception in normal reception

```

typedef struct
{
    r_a_as_req_err_t result;      Result of transmission and reception
                                See the enumerated type "4.14.3(1) r_a_as_req_err_t" for
                                details.

    r_a_as_data_t   data;        Reception data
                                See the structure "4.14.1(5) r_a_as_data_t" for details.

    r_a_as_status_t status;      State of the A_AS
                                See the structure "4.14.1(6) r_a_as_status_t" for details.

} r_a_as_result_t

```

(5) r_a_as_data_t

Data received in normal reception

```
typedef struct
{
    uint32_t    rxi;           RXI register value
                               The value of the RXI register is stored here.
    uint32_t    rxd0;        RXD0 register value
                               The value of the RXD0 register is stored here.
    uint32_t    rxd1;        RXD1 register value
                               The value of the RXD1 register is stored here.
} r_a_as_data_t
```

(6) r_a_as_status_t

State of the A_AS on transmission or reception

```
typedef struct
{
    bool        iwdgerr;      IF watchdog errors. (true: occurred; false: not occurred)
    bool        dwdgerr;     DF watchdog errors. (true: occurred; false: not occurred)
    bool        starterr;    Start bit errors. (true: occurred, false: not occurred)
    bool        stoperr;     Stop bit errors. (true: occurred, false: not occurred)
    bool        syncerr;     Sync code errors. (true: occurred; false: not occurred)
    bool        rxearr;      Received encoder address errors. (true: occurred; false: not occurred)
    bool        crcerr;      CRC errors. (true: occurred; false: not occurred)
    bool        rxccerr;     Received command code errors. (true: occurred; false: not occurred)
    bool        mdaterr;     EEPROM data errors. (true: occurred; false: not occurred)
    bool        madrerr;     EEPROM address errors. (true: occurred; false: not occurred)
    bool        rxdzerr;     Recognition code errors. (true: occurred; false: not occurred)
    bool        fd1err;      Fixed-data (1) errors. (true: occurred; false: not occurred)
    bool        fd2err;      Fixed-data (2) errors. (true: occurred; false: not occurred)
    bool        fd3err;      Fixed-data (3) errors. (true: occurred; false: not occurred)
    bool        fd5err;      Fixed-data (5) errors. (true: occurred; false: not occurred)
    bool        fd6err;      Fixed-data (6) errors. (true: occurred, false: not occurred)
    bool        fd7err;      Fixed-data (7) errors. (true: occurred, false: not occurred)
    bool        bankerr;     Bank errors. (true: occurred, false: not occurred)
    bool        elcin;       ELCIN input information is stored.
                               (true: Data transfer has started by ELC event trigger input.)
    uint8_t     txcc;        Transmission command code is stored.
                               (0: CDF0 to 20 and 23 to 30, 1: CDF21, 2: CDF22)
    bool        rxset;       Whether the received data is ready. (true: ready; false: not ready)
    bool        timer;       Timer status information (true: operating; false: stopped)
    bool        txerr;       Transmission errors. (true: occurred; false: not occurred)
    bool        rxend;       Completion of reception. (true: completed; false: not completed)
} r_a_as_status_t
```

4.14.2 A-format Unions

Unions are not used in this sample program.

4.14.3 A-format Enumerated Types

(1) r_a_as_req_err_t

Result of reception from the encoder

```
typedef enum
{
    R_A_AS_REQ_SUCCESS = 0,    Normal termination of data transmission and reception
    R_A_AS_REQ_ERR            An error occurred in data transmission or reception.
                              This error code is generated even if one of the error indicators
                              (excluding timer, rxend, rxset, elcin, and txcc) of the structure
                              "4.14.1(6) r_a_as_status_t" is "true".

    R_A_AS_REQ_BP_ERR        The reception FIFO buffer is full.
                              This error code is generated only in bypass reception.
} r_a_as_req_err_t
```

4.14.4 EnDat Structures

(1) r_endat_info_t

Initialization information of the EnDat control unit

```
typedef struct
{
    uint8_t      ftclk;           Transmission clock frequency setting
                                See "Table 4.18 EnDat Transmission Clock
                                Frequencies". This setting is reflected in the FTCLK bit
                                of the CFG1 register.

    bool         filter;         Noise filter settings (true: enabled, false: disabled)
                                This setting is reflected in the INF bit, NFINF bit, and
                                NFSCNT bit of the NF register.

    bool         delay_comp;     Propagation delay correction (true: valid, false: invalid)
                                This setting is reflected in the DLY bit of the CFG1
                                register.

    uint8_t      tst;           Set the Low period at the start of data transmission
                                See "Table 4.20 EnDat Low-level Period at the Start of
                                Data Transmission". This setting is reflected in the TST
                                bit of the CFG2 register.

    endat_wait_cb_t p_enc_init_reset_wait; A pointer to a callback function that generates the wait
                                time after an encoder reset
                                See "4.9.1 enc_init_reset_wait_callback" for details.
                                Do not set NULL.

    endat_wait_cb_t p_enc_init_mem_wait;  Pointer to callback function that generates wait time for
                                encoder memory area selection timeout error detection.
                                See "4.9.2 enc_init_mem_wait_callback" for details.
                                Do not set NULL.

    endat_wait_cb_t p_enc_init_pram_wait; Encoder parameter Send / receive timeout Error
                                detection function pointer to generate wait time
                                See "4.9.3 enc_init_pram_wait_callback" for details.
                                Do not set NULL.

    endat_wait_cb_t p_enc_init_cable_wait; A pointer to a function that produces a wait time for
                                time-out error detection for cable propagation delay
                                measurements. If propagation delay compensation is
                                disabled (delay_comp = false), the setting can be
                                omitted.
                                See "4.9.4 enc_init_cable_wait_callback" for details.
                                Do not set NULL when propagation delay compensation
                                is enabled.

} r_endat_info_t
```

(2) r_endat_watchdog_t

Watchdog Timer setting time

```

typedef struct
{
    uint8_t          range;    Set the unit of time for the Watchdog Timer
                        See "Table 4.19 EnDat Watchdog Timer Time Units"
    uint8_t          time;    Set the Watchdog Timer time
                        See "Table 4.23 EnDat Watchdog Timer Table"
} r_endat_watchdog_t

```

Table 4.23 EnDat Watchdog Timer Table

Value of time	Time of the Watchdog Timer	
	range = R_ENDAT_WD_RANGE_US	range = R_ENDAT_WD_RANGE_MS
0	Stop	Stop
1	2 us	0.2 ms
2	4 us	0.4 ms
3	6 us	0.6 ms
:	:	:
10	20 us	2.0 ms
:	:	:
127	254 us	25.4 ms

Note: Except for the stop time, there is a margin of error. Refer to the hardware manual for details.

(3) r_endat_req_t

Request information to be sent to the EnDat2.2 compliant encoder. The mode command, MRS code, address and port address are combined and sent to the encoder. The combinations are shown in "Table 4.23 Mode Command Combination Table".

```

typedef struct
{
    uint8_t          mode_cmd;    EnDat 2.2 Mode Command
                                See "Table 4.17 EnDat 2.2 Mode Commands".
    bool             dt;         Continuous mode setting (true: enabled, false:
                                disabled)
                                This setting is valid only if mode_cmd =
                                R_ENDAT_POS.
    uint8_t          mrs;       MRS code
                                See "Table 4.21 EnDat MRS Codes".
                                The setting is valid only if the mode command
                                combined with the MRS code in the "Table 4.24 Mode
                                Command Combination Table" is designated as the
                                mode_cmd.
    uint8_t          addr;      Address (0x00 to 0xFF)
                                The setting is valid only if the mode command
                                combined with the address in the "Table 4.24 Mode
                                Command Combination Table" is designated as the
                                mode_cmd.
    uint16_t         param_instruction; Parameters to be written to memory area of the
                                encoder
                                The setting is valid only if the mode command
                                combined with the parameters or block address in the
                                "Table 4.24 Mode Command Combination Table" is
                                designated as the mode_cmd.
    r_endat_watchdog_t watchdog; Setting time of watchdog timer
                                See "4.14.4(2) r_endat_watchdog_t".
                                When sending a request for the following settings, set
                                it to disabled (time =0).
                                    mode_cmd=R_ENDAT_POS and dt=true
                                    mode_cmd=R_ENDAT_RESET
                                    mode_cmd=R_ENDAT_RX_PARAM
                                    mode_cmd=R_ENDAT_PARAM
                                This setting is reflected in the CFG2 register WDG bit.
    bool             elc;       ELC mode setting (true: enabled, false: disabled)
                                This setting is valid only if
                                mode_cmd=R_ENDAT_POS and dt=false.
    r_endat_isr_result_cb_t p_isr_result; Pointer to a callback function that conveys the result
                                of the request.
                                See "4.9.5 endat_callback" and "4.9.6
                                endat_poscon_callback" for details.
                                Do not set NULL.
    r_endat_isr_rdst_cb_t p_isr_rdst;  Pointer to a callback function that conveys that the
                                next data communication is ready.
                                See "4.9.7 endat_rdst_callback" for detail.
                                Do not set NULL.
} r_endat_req_t

```

Table 4.24 Mode Command Combination Table

mode_cmd	Command value	mrs / addr	param_instruction
R_ENDAT_POS	0x07u	--	--
R_ENDAT_MEM	0x0Eu	MRS Code	--
R_ENDAT_RX_PARAM	0x1Cu	Address	Parameters *1
R_ENDAT_PARAM	0x23u	Address	--
R_ENDAT_RESET	0x2Au	Address	--
R_ENDAT_POS_ADD_DATA	0x38u	--	--
R_ENDAT_POS_MEM	0x09u	MRS Code	Block address *2
R_ENDAT_POS_RX_PARAM	0x1Bu	Address	Parameters *1
R_ENDAT_POS_PARAM	0x24u	Address	--
R_ENDAT_POS_RX_ERR_RESET	0x2Du	Address	--

Note 1. Consider the setting value according to the address.

2. Use only when the MRS code is R_ENDAT_MRS_PARAM_SEC2

(4) r_endat_result_t

Send/receive results

```
typedef struct
{
    r_endat_req_err_t    result;    Results of sending and receiving requests
                               See "4.14.6(3) r_endat_req_err_t".
    r_endat_data_t      data;      Received data
                               See "4.14.4(5) r_endat_data_t".
    r_endat_status_t    status;    Encoder Status
                               See "4.14.4(6) r_endat_status_t".
} r_endat_result_t
```

(5) r_endat_data_t

Received data

```
typedef struct
{
    uint64_t            pos;        Received positional value or test value
                               The RXD1 bit in the RECV1L register is stored in the
                               lower 32 bits. The RXD1 bit in the RECV1U register is
                               stored in the upper 32 bits.
    Uint32_t            add_datum1; Additional data 1
                               Stores the RXD3-bit value of the RECV3 register.
    Uint32_t            add_datum2; Additional data 2
                               Stores the RXD2 bit value of the RECV2 register.
} r_endat_data_t
```

(6) r_endat_status_t

Encoder Status

```

typedef struct
{
    bool        busy;        Encoder internal memory status
                        (true: accessing, false: accessible)
    bool        rm;         Increment encoder origin status
                        (true: origin detection, false: origin undetected)
    bool        wrn;        Warning status inside the encoder
                        (true: with warning, false: no warning)
} r_endat_status_t

```

(7) r_endat_protocol_err_t

EnDat I/F and encoder error information

```

typedef struct
{
    bool        err1;       Error1 bit status (true: occurred, false: not occurred)
    bool        crc1;       CRC check error for positional value (true: occurred, false: not
                        occurred)
    bool        ftype1;     EnDat TYPE1 error (true: occurred, false: not occurred)
    bool        ftype2;     EnDat TYPE2 error (true: occurred, false: not occurred)
    bool        msadr;      Address error in EnDat TYPE2 error (true: occurred, false: not
                        occurred)
    bool        err2;       Error2 bit status (true: occurred, false: not occurred)
    bool        crc3;       CRC check error for Additional data 1 (true: occurred, false: not
                        occurred)
    bool        crc2;       CRC check error for Additional data 2 (true: occurred, false: not
                        occurred)
    bool        wdg;        Watchdog error (true: occurred, false: not occurred)
    bool        ftype3;     EnDat TYPE3 error (true: occurred, false: not occurred)
    bool        modeerr;    Mode command transmission error (true: occurred, false: not
                        occurred)
} r_endat_protocol_err_t

```

4.14.5 EnDat Unions

No unions are used.

4.14.6 EnDat Enumerated Types**(1) r_endat_err_t**

Error codes of the encoder I/F

```

typedef enum
{
    ENDAT_SUCCESS           =0,    Normal termination
    ENDAT_ERR_INVALID_ARG   Argument error
    ENDAT_ERR_BUSY         API is not executable
    ENDAT_ERR_ACCESS       Error in the execution order of APIs
    ENDAT_ERR_DRV          Internal error in driver
} r_endat_err_t

```

(2) r_endat_cmd_t

Command settings when the R_ENDAT_Control function is used

```

typedef enum
{
    ENDAT_CMD_REQ           Send command to the encoder
    ENDAT_CMD_POS_STOP     Stop continuous reception of positional values from the
                           encoder
} r_endat_cmd_t

```

(3) r_endat_req_err_t

Result of sending and receiving requests

```

typedef enum
{
    ENDAT_REQ_SUCCESS      =0,    Normal completion of data transmission and reception
    ENDAT_REQ_ERR         Data transmission/reception control error occurs
} r_endat_rx_err_t

```

4.15 Description of the Sample Program

4.15.1 Operation Outline

This sample program supports bus connection of up to eight encoders compliant with the A-format specification (Nikon "MAR-M50A" or "SAR-HL700A") and the EnDat 2.2 compliant encoder "EQN1035". This sample program performs the following processes.

A-format

- 1) Request information input from the console is sent to the encoder (normal sending/receiving by writing to the TRG register).
- 2) Display data received from the encoder on the console.
- 3) Send and receive commands using the ELC event input trigger function of A_AS. (GPT events are linked as input events. For an example of input event settings, see "Figure 4.6 Flowchart of a_as_elctimer function.")

EnDat

- 1) Send requests input from the debugger's terminal I/O to the EnDat encoder (EQN1035).
- 2) Display data received from the EnDat encoder (EQN1035) on the debugger's terminal I/O.
- 3) Send and receive commands using the ELC event input trigger function of the EnDat I/F. (GPT events are linked as input events.)

(1) Dual Encoder System Block Diagram

Figure 4.1 shows a block diagram of the system.

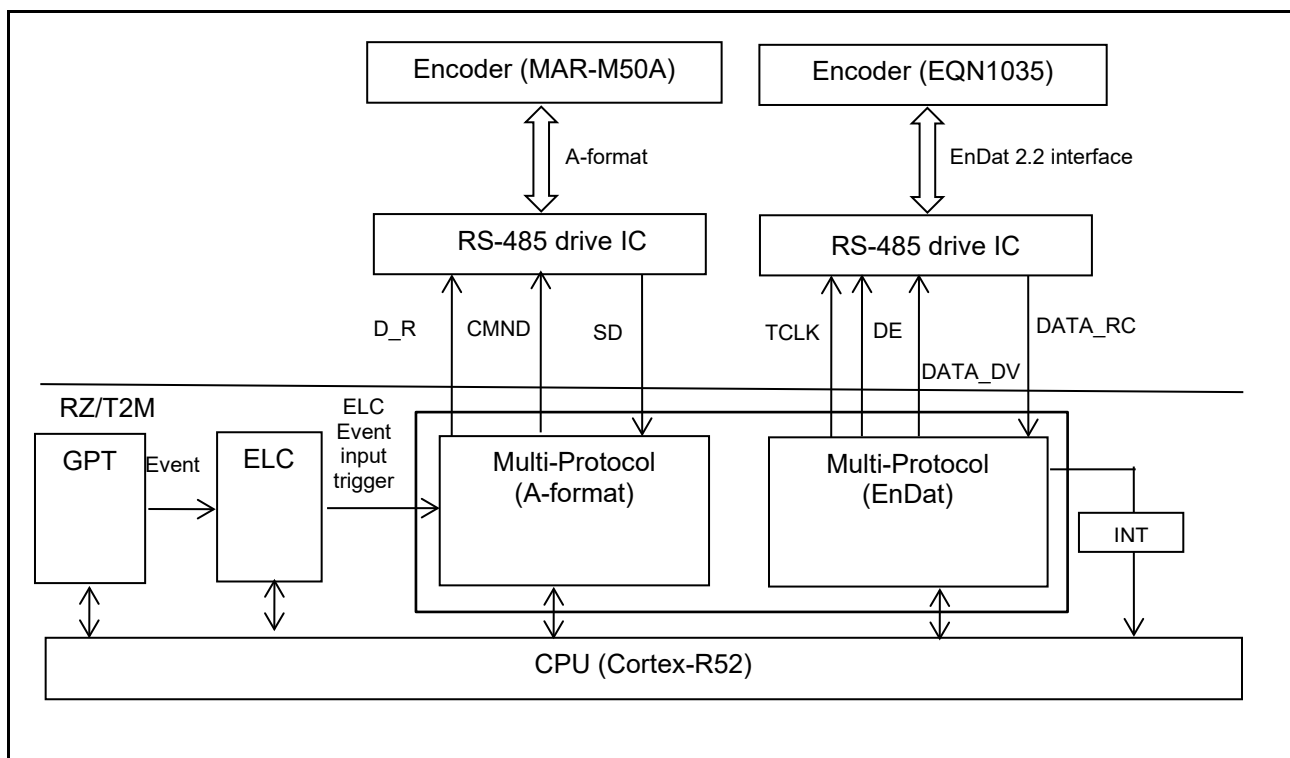


Figure 4.1 System Block Diagram

(2) Software Structure

Figure 4.2 shows the structure of the software.

The A-format driver has the opening process part consisting of the R_A_AS_Open function, the closing process part consisting of the R_A_AS_Close function, the sending requests part consisting of the R_A_AS_Control function, and the data reception part (interrupt handler) consisting of callback functions.

The EnDat driver has the opening process part consisting of the R_ENDAT_Open function, the closing process part consisting of the R_ENDAT_Close function, the sending requests part consisting of the R_ENDAT_Control function, and the data reception part (interrupt handler) consisting of callback functions.

The sample program has the A-format driver control part that controls the A-format driver and sends requests, and the A-format result indication part (callback) that displays the results of data reception, the EnDat driver control part that controls the EnDat driver and sends requests, and the EnDat result indication part (callback) that displays the results of data reception.

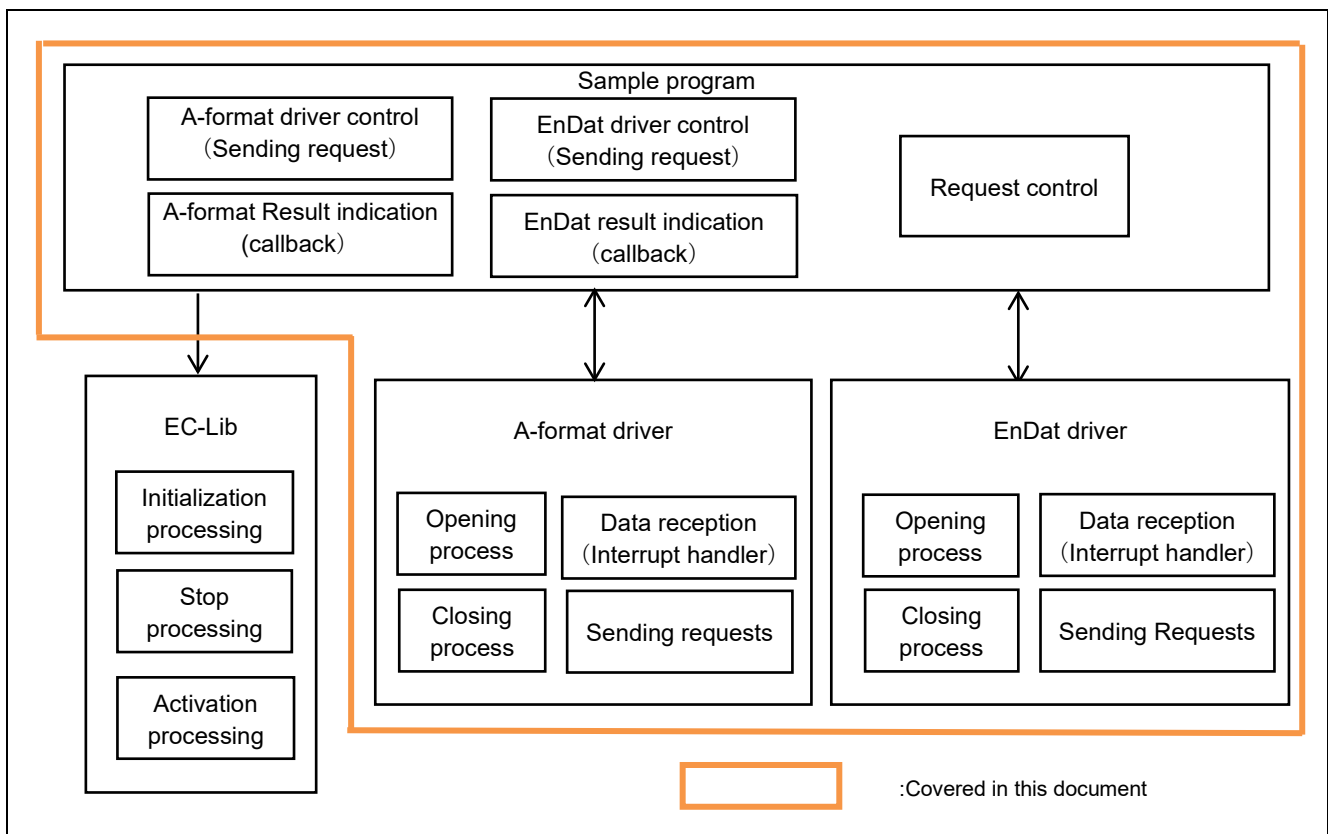


Figure 4.2 Dual Encoder Software Structure

4.15.2 Sample Program Functions

Table 4.25 lists the major functions of the sample program.

Table 4.25 Major Functions of the Sample Program

Function Type	Function Name	Page Number	
		Specification	Flowchart
Common Function	hal_entry	49	-
	enc_main	49	60
	dual_cmd_control	49	61
	get_cmd	49	-
	cmd_exit	50	-
	timer_start	50	-
	timer_stop	50	-
A-format Function	a_as_enc_init	50	-
	a_as_req	51	62
	a_as_setdf	51	63
	a_as_elctimer	51	64
	a_as_elcstop	51	65
	a_as_txerr_callback	52	65
	a_as_rxset_callback	52	66
	a_as_rxend_callback	52	66
	a_as_elctimer_callback	53	67
EnDat Function	endat_power_on_wait	53	-
	enc_init_reset_wait_callback	53	-
	enc_init_mem_wait_callback	53	-
	enc_init_pram_wait_callback	54	-
	enc_init_cable_wait_callback	54	-
	endat_pos	54	68
	endat_poscon	54	69
	endat_elctimer	55	70
	endat_stop	55	71
	endat_temp	55	72
	endat_callback	55	73
	endat_poscon_callback	56	74
	endat_rdst_callback	56	74
	endat_result_display	56	-

4.15.3 Specifications of Sample Program Functions

(1) Common Functions

(a) hal_entry

hal_entry	
Synopsis	Entry function of the Dual Encoder sample program
Header	-
Declaration	void hal_entry(void);
Description	This is the entry function of the Dual Encoder sample program. From here, the function enc_main () is called.
Arguments	None
Return value	None

(b) enc_main

enc_main	
Synopsis	Main function of the Dual Encoder sample program
Header	-
Declaration	int32_t enc_main(void);
Description	This is the main function of the Dual Encoder sample program. For details, see section "4.15.8(1) Flowchart of enc_main".
Arguments	None
Return value	0 : Normal termination Others : Abnormal termination (error code of the encoder interface)

(c) dual_cmd_control

dual_cmd_control	
Synopsis	Dual Encoder driver control function
Header	-
Declaration	static void dual_cmd_control(void);
Description	This function performs console command input processing.
Arguments	None
Return value	None

(d) get_cmd

get_cmd	
Synopsis	Function to get command from console
Header	-
Declaration	static uint32_t get_cmd(char_t *p_arg[], const uint32_t arg_max);
Description	Get the command from the console
Arguments	p_arg : Pointer to an array console that stores commands retrieved from the console arg_max : Maximum number of strings to retrieve
Return value	Number of strings retrieved from console

(e) cmd_exit

cmd_exit	
Synopsis	End display function of the Dual Encoder sample program
Header	-
Declaration	static void cmd_exit(uint32_t arg_num, char_t *p_arg[]);
Description	Function executed when the console command exit is entered, indicating on the console that the sample program has terminated.
Arguments	arg_num : Number of strings entered from console (not used) *p_arg[] : First address of string entered from console (not used)
Return value	None

(f) timer_start

timer_start	
Synopsis	GPT cycle setting/startup function
Header	-
Declaration	static void timer_start(uint32_t ch, uint32_t us);
Description	Set the timer cycle in GPT corresponding to the encoder of designated channel to start the timer.
Arguments	ch : Encoder channel number us : Timer cycle [us]
Return value	None

(g) timer_stop

timer_stop	
Synopsis	GPT timer stop
Header	-
Declaration	static void timer_stop(uint32_t ch);
Description	Stop the GPT timer corresponding to the encoder of designated channel.
Arguments	ch : Encoder channel number
Return value	None

(2) A-format Functions**(a) a_as_enc_init**

a_as_enc_init	
Synopsis	Initializing the encoder
Header	-
Declaration	static int32_t a_as_enc_init(int32_t id);
Description	This function is for initializing the encoder. The command data frame CDF8 is transmitted eight times consecutively to clear the status flag.
Argument	id : Encoder ID R_A_AS0_ID: Specify Encoder ID ch0 R_A_AS1_ID: Specify Encoder ID ch1
Return value	0: Normal termination Others: Abnormal termination (error code of the encoder interface)

(b) a_as_req

a_as_req	
Synopsis	Console command "req" function
Header	-
Declaration	static void a_as_req(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command "req" is input. See section "4.15.8(3) Flowchart of a_as_req" and section "4.15.10 Console Commands" for details.
Argument	arg_num : The number of character strings input through the console. *p_arg[] : The starting address where the character strings are stored.
Return value	None

(c) a_as_setdf

a_as_setdf	
Synopsis	Console command "setdf" function
Header	-
Declaration	static void a_as_setdf(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command "setdf" is input. See section "4.15.8(4) Flowchart of a_as_setdf" and "4.15.10 Console Commands" for details.
Argument	arg_num : The number of character strings input through the console. *p_arg[] : The starting address where the character strings are stored.
Return value	None

(d) a_as_elctimer

a_as_elctimer	
Synopsis	Console command "elctimer" function
Header	-
Declaration	static void a_as_elctimer (uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command "elctimer" is input. See section "4.15.8(5) Flowchart of a_as_elctimer" and section "4.15.10 Console Commands" for details.
Arguments	arg_num : The number of character strings input through the console. *p_arg[] : The starting address where the character strings are stored.
Return value	None

(e) a_as_elcstop

a_as_elcstop	
Synopsis	Console command "elcstop" function
Header	-
Declaration	static void a_as_elcstop (uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command "elcstop" is input. See section "4.15.8(6) Flowchart of a_as_elcstop" and section "4.15.10 Console Commands" for details.
Argument	arg_num : The number of character strings input through the console. *p_arg[] : The starting address where the character strings are stored.
Return value	None

(f) a_as_txerr_callback**a_as_txerr_callback**

Synopsis	Callback for the console command "req" when a transmission error is generated
Header	-
Declaration	static void a_as_txerr_callback(r_a_as_result_t *p_result);
Description	This is a callback function that is executed when the console command "req" is input. It holds a pointer to the result of transmitted A-format request in normal reception in the variable a_as_result and indicates that a transmission error is generated. See section "4.15.8(7) Flowchart a_as_txerr_callback" for details.
Argument	*p_result : The address in the RAM where the result of transmission of the request and reception starts.
Return value	None

(g) a_as_rxset_callback**a_as_rxset_callback**

Synopsis	Callback for the console command "req" when the setting of received data is complete.
Header	-
Declaration	static void a_as_rxset_callback(r_a_as_result_t *p_result);
Description	This is a callback function that is executed when the console command "req" is input. It holds a pointer to the result of the transmitted an A-format request and received data in response in normal reception in the variable a_as_result and indicates completion of the setting of received data. See section "4.15.8(8) Flowchart of a_as_rxset_callback" for details.
Argument	*p_result : The address in the RAM where the result of transmission of the request and reception starts.
Return value	

(h) a_as_rxend_callback**a_as_rxend_callback**

Synopsis	Callback for the console command "req" when reception of the data has been completed.
Header	-
Declaration	static void a_as_rxend_callback(r_a_as_result_t *p_result);
Description	This is a callback function that is executed when the console command "req" is input. It indicates completion of the reception of data in response to the A-format request in normal reception. See section "4.15.8(9) Flowchart of a_as_rxend_callback" for details.
Argument	*p_result : The address in the RAM where the result of transmission of the request and reception starts.
Return value	None

(i) a_as_elctimer_callback**a_as_elctimer_callback**

Synopsis	Callback for the console command "elctimer".
Header	-
Declaration	static void a_as_elctimer_callback(r_a_as_result_t * p_result);
Description	This is a callback function that is executed when the console command "elctimer" is input. It holds the result of the transmitted request and received data in the variables a_as_ti_result and a_as_ti_data. See section "4.15.8(10) Flowchart of a_as_elctimer_callback" for details.
Argument	*p_result : The address in the RAM where the result of transmission of the request and reception starts.
Return value	None

(3) EnDat functions**(a) endat_power_on_wait****endat_power_on_wait**

Synopsis	Waiting time generation function after encoder power-on
Header	-
Declaration	static void endat_power_on_wait(void);
Description	This callback function generates the required 1.3s standby time after the encoder is turned on.
Arguments	None
Return value	None

(b) enc_init_reset_wait_callback**enc_init_reset_wait_callback**

Synopsis	Waiting time generation function after encoder reset
Header	-
Declaration	static void enc_init_reset_wait_callback(void);
Description	This callback function generates a waiting time of 60 ms after the encoder reset process in the initialization process of the connected encoder. See "4.9.1 enc_init_reset_wait_callback".
Arguments	None
Return value	None

(c) enc_init_mem_wait_callback**enc_init_mem_wait_callback**

Synopsis	Waiting time generation function for encoder memory area selection process
Header	-
Declaration	static void enc_init_mem_wait_callback(void);
Description	This callback function generates a waiting time of 743 us for detecting a timeout error in the process of selecting a memory area in the initialization process of the connected encoder. See "4.9.2 enc_init_mem_wait_callback".
Arguments	None
Return value	None

(d) enc_init_pram_wait_callback

enc_init_pram_wait_callback	
Synopsis	Waiting time generation function for encoder parameter sending/receiving process
Header	-
Declaration	static void enc_init_pram_wait_callback(void);
Description	This callback function generates a waiting time of 13 ms for detecting a timeout error in the initialization process of the connected encoder, during which the encoder sends and receives parameters. See "4.9.3 enc_init_pram_wait_callback"
Arguments	None
Return value	None

(e) enc_init_cable_wait_callback

enc_init_cable_wait_callback	
Synopsis	Wait time generation function for encoder cable propagation delay measurement process
Header	-
Declaration	static void enc_init_cable_wait_callback(void);
Description	Callback function to generate a waiting time of 588 us for detecting a timeout error in the process of measuring the cable propagation delay in the initialization process of the connected encoder. See "4.9.4 enc_init_cable_wait_callback"
Arguments	None
Return value	None

(f) endat_pos

endat_pos	
Synopsis	Function to get a positional value from the encoder
Header	-
Declaration	static void endat_pos(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command pos is entered. It acquires a positional value from the encoder.
Arguments	arg_num : Number of strings entered from the console (Not used) *p_arg[] : First address of string entered from console (Not used)
Return value	None

(g) endat_poscon

endat_poscon	
Synopsis	Function to get positional values continuously from the encoder
Header	-
Declaration	static void endat_poscon(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command poscon is entered to continuously acquire position values from the encoder using Continuous mode.
Arguments	arg_num : Number of strings entered from the console (Not used) *p_arg[] : First address of string entered from console (Not used)
Return value	None

(h) endat_elctimer

endat_elctimer	
Synopsis	Function to get positional values continuously from the encoder synchronously with the ELC events
Header	-
Declaration	static void endat_elctimer(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command elctimer is entered. It acquires positional values continuously from the encoder synchronously with the ELC events.
Arguments	arg_num : Number of strings entered from the console *p_arg[] : First address of string entered from console
Return value	None

(i) endat_stop

endat_stop	
Synopsis	Function to stop continuous acquisition positional values from the encoder
Header	-
Declaration	static void endat_stop(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command stop is entered. It stops the continuous transmission of positional values from the encoder when it is operating in Continuous mode. While operating in ELC mode, this function cancels ELC mode operation and stops issuing continuous positional value acquisition commands. After the continuous positional value transmission from the encoder is stopped, the last 10 positional values are displayed.
Arguments	arg_num : Number of strings entered from the console (Not used) *p_arg[] : First address of string entered from console (Not used)
Return value	None

(j) endat_temp

endat_temp	
Synopsis	Function to get temperature information from the encoder
Header	-
Declaration	static void endat_temp(uint32_t arg_num, char_t *p_arg[]);
Description	This function is executed when the console command temp is entered. It acquires temperature information from the encoder
Arguments	arg_num : Number of strings entered from the console (Not used) *p_arg[] : First address of string entered from console (Not used)
Return value	None

(k) endat_callback

endat_callback	
Synopsis	Callback function that conveys the result of the request transmission to the encoder
Header	-
Declaration	static void endat_callback(r_endat_result_t *p_result, r_endat_protocol_err_t *p_err);
Description	This function stores the result in memory.
Arguments	p_result : Result of the request transmission p_err : EnDat I/F and encoder error information
Return value	None

(l) endat_poscon_callback**endat_poscon_callback**

Synopsis	Callback function that conveys the result of the request transmission to the encoder
Header	-
Declaration	static void endat_poscon_callback (r_endat_result_t *p_result, r_endat_protocol_err_t *p_err);
Description	This function stores the continuously acquired results in memory
Arguments	p_result : Result of the request transmission p_err : EnDat I/F and encoder error information
Return value	None

(m) endat_rdst_callback**endat_rdst_callback**

Synopsis	Callback function to convey that the next data transmission is ready to start
Header	-
Declaration	static void endat_rdst_callback(void);
Description	This function conveys that the data reception completes and the next data communication is ready. It is called each time data reception is completed while operating in continuous mode or in ELC mode. This function sets the acquisition completion flag.
Arguments	None
Return value	None

(n) endat_result_display**endat_result_display**

Synopsis	Function to display the result of data reception
Header	-
Declaration	static void result_display(r_endat_result_t *p_result, r_endat_protocol_err_t *p_err);
Description	This function indicates the result of data reception in response to a request sent to the encoder.
Arguments	p_result : Result of the request transmission p_err : EnDat I/F and encoder error information
Return value	None

4.15.4 Variables Used in the A-format Sample Program

The major static variables used in the sample program are listed below.

Table 4.26 Static Variables Used in the A-format Sample Program

Type	Variable Name	Description
bool	a_as_flg	Transmission and reception completion flag. (true: transmission and reception completed; false: transmission and reception in progress)
r_a_as_result_t	a_as_result[A_AS_ENC_NUM]	The results of acquisition are stored.
bool	a_as_elc_flg	ELC event input trigger flag (true: ELC event input trigger operation is in progress; false: ELC event input trigger operation is stopped)
bool	elc_trans_flg	The flag that indicates the state of transmission and reception of data from the ELC event input trigger. (true: Transmission and reception in progress, false: Transmission and reception completed)
r_a_as_req_t	a_as_req_elc	Holds the request information while ELC event input trigger operation is in progress.
uint8_t	enc_df_type[A_AS_ENC_NUM]	Data frame types for CDF1, CDF5, and CDF8 to CDF12 of the A-format version 3.0 encoders or later are stored.

4.15.5 Constants Used in the A-format Sample Program

The major constants used in the sample program are listed below.

Table 4.27 Major Constants Used in the A-format Sample Program

Constant	Setting	Description
A_AS_ENC_NUM	8	The number of connected encoders.

4.15.6 Variables of EnDat Sample Program

Table 4.28 lists the major static type variables. Const type variables are not used.

Table 4.28 Static Variables Used in the EnDat Sample Program

Type	Variable Name	Description	Function to be used
bool	endat_flg	Transmission completion flag (true: transmission completed, false: transmission is in progress)	endat_pos endat_poscon endat_elctimer endat_stop endat_temp endat_callback endat_rdst_callback
bool	endat_elc_flg	ELC mode operating flag (true: operating in ELC mode, false: not operating in ELC mode)	endat_pos endat_poscon endat_elctimer endat_stop
r_endat_result_t	*p_endat_result	Address containing data acquisition results	endat_pos endat_temp endat_callback
r_endat_protocol_err_t	*p_endat_err	Address containing error information	endat_pos endat_temp endat_callback
r_endat_req_err_t	poscon_err[ENDAT_POS_NUM]	Errors in continuously acquired positional values An array with 10 elements is used as a ring buffer to store the results of the latest 10 acquisitions.	endat_poscon endat_elctimer endat_stop endat_poscon_callback
uint64_t	poscon[ENDAT_POS_NUM]	Continuously acquired positional values An array with 10 elements is used as a ring buffer to store the results of the latest 10 acquisitions.	endat_poscon endat_elctimer endat_stop endat_poscon_callback
uint8_t	poscon_valid	Number of valid elements in poscon, poscon_err array Indicates the number of valid elements of positional values stored in the array.	endat_poscon endat_elctimer endat_stop endat_poscon_callback
uint8_t	poscon_num	Update position indices for poscon and poscon_err arrays The following are the indexes to be updated by the acquired position values.	endat_poscon endat_elctimer endat_stop endat_poscon_callback
bool	poscon_empty	Space information in poscon and poscon_err arrays (true: has space, false: has no space)	endat_poscon endat_elctimer endat_poscon_callback
int32_t	cur_id	Used EnDat I/F driver ID	enc_main endat_cmd_control endat_pos endat_poscon endat_elctimer endat_stop endat_temp

4.15.7 Constants of EnDat Sample Program

Table 4.29 lists the major constants used in the sample program.

Table 4.29 Major Constants Used in the EnDat Sample Program

Constants	Value	Content
ENDAT_ENC_TSAT_WAIT	1300u	Standby time after power-on (1.3 s)
ENDAT_ENC_100US_WAIT	100u	Waiting time after EC-Lib startup (100 us)
ENDAT_ENC_INIT_RESET_WAIT	60u	Time to wait after encoder reset process (60 ms)
ENDAT_ENC_INIT_MEM_WAIT	743u	Waiting time for detection of timeout errors in the process of selecting memory area in encoder initialization (743 us)
ENDAT_ENC_INIT_PRAM_WAIT	13u	Waiting time for detection of timeout errors in the process of sending and receiving parameters in encoder initialization. (13 ms)
ENDAT_ENC_INIT_CABLE_WAIT	588u	Waiting time for detection of timeout errors in the process of measuring cable propagation delay in encoder initialization. (588 us)
ENDAT_WDG_MAX	127u	Maximum watchdog timer setting value
ENDAT_POS_NUM	10u	Number of elements of the array for storing continuously received position values
ENDAT_TEMP_SCA_FAC	0.1	Number of elements of the array for storing continuously received position values
ENDAT_TEMP_ABS_ZERO	273.2	Constant for temperature data unit conversion

4.15.8 Flowchart of Main Processing

(1) Flowchart of enc_main

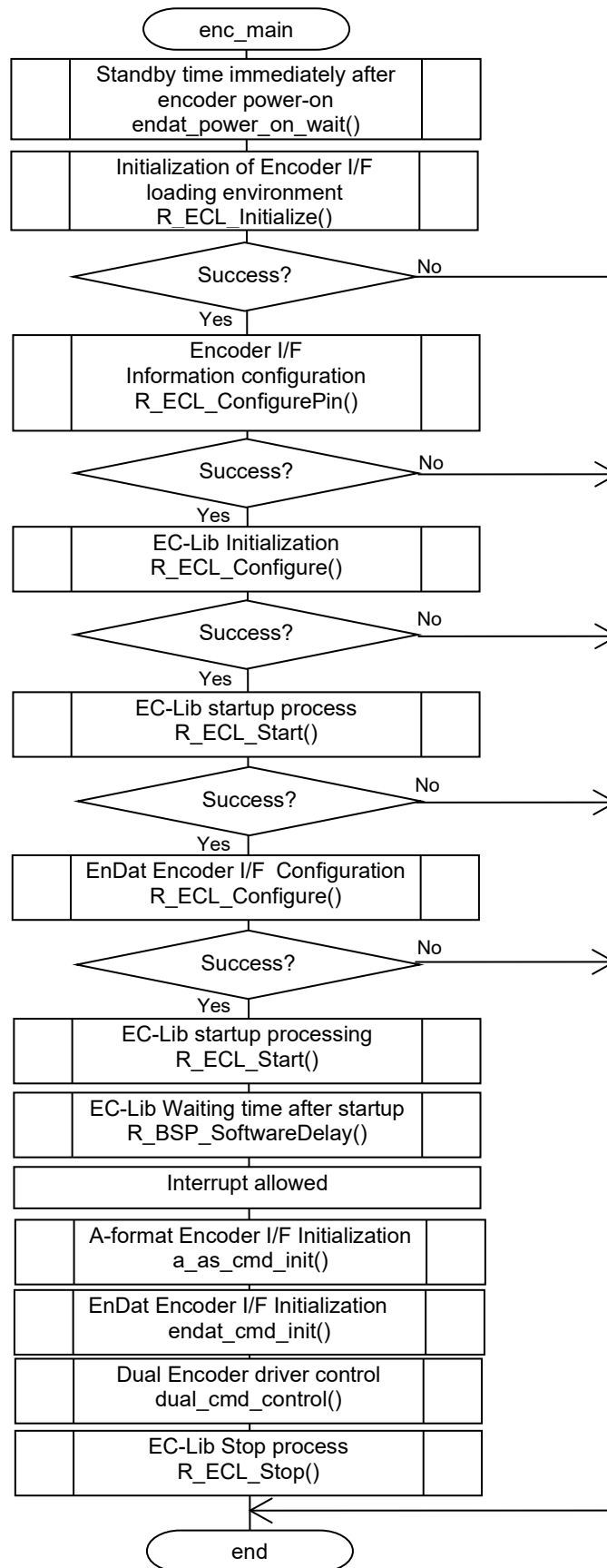


Figure 4.3 Flowchart of enc_main Function

(2) Flowchart of dual_cmd_control

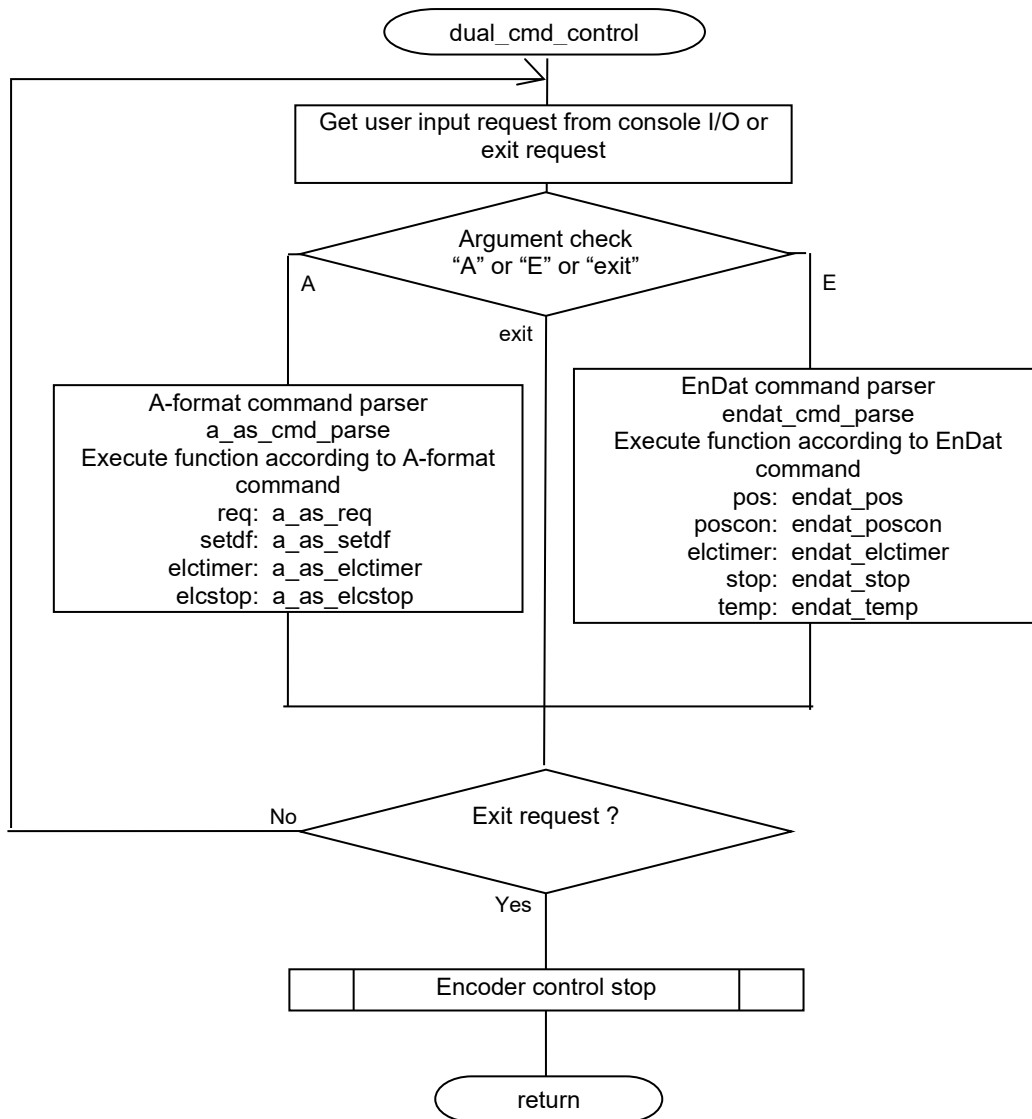


Figure 4.4 Flowchart of dual_cmd_control Function

(3) Flowchart of a_as_req

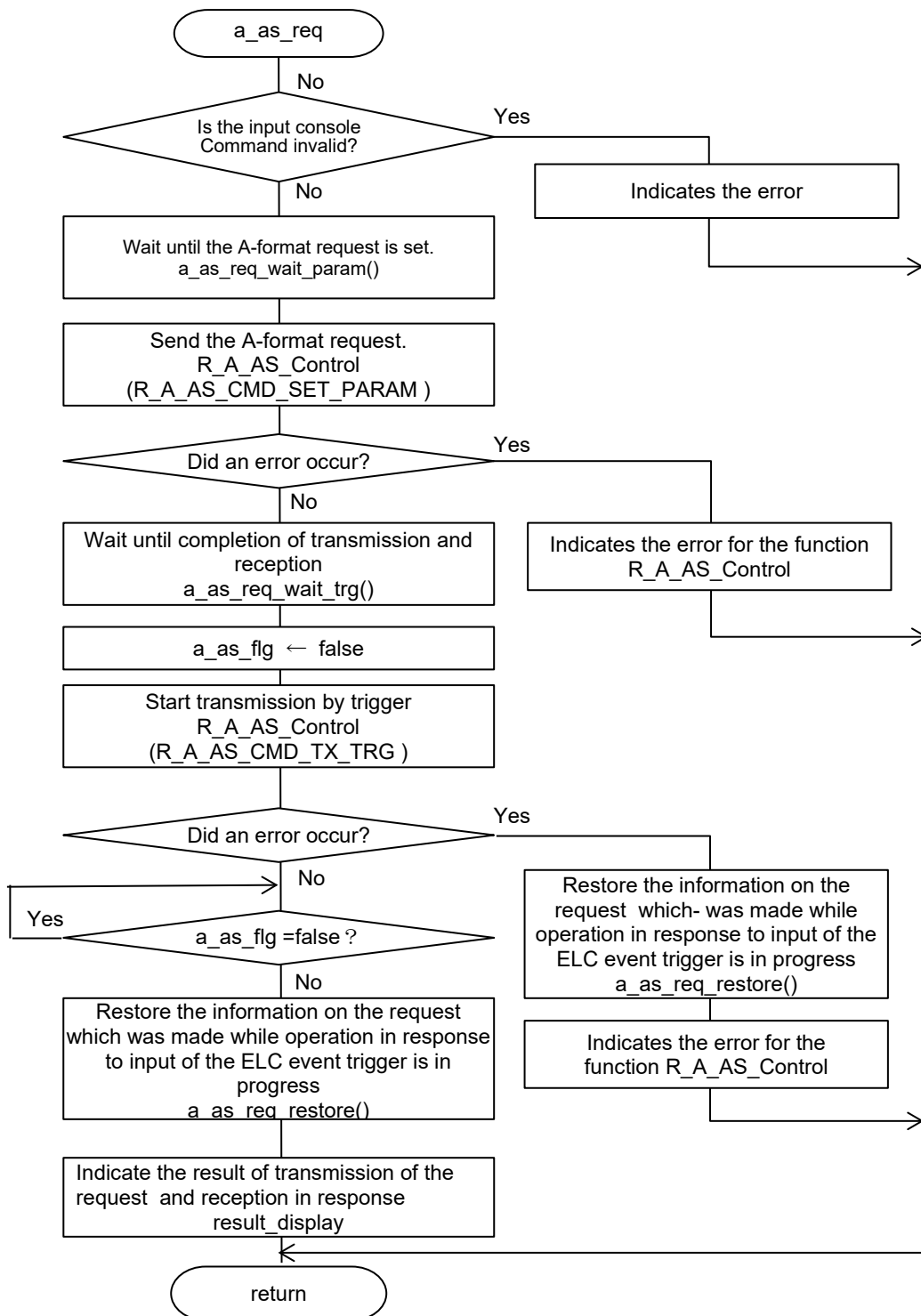


Figure 4.5 Flowchart of a_as_req Function

(4) Flowchart of a_as_setdf

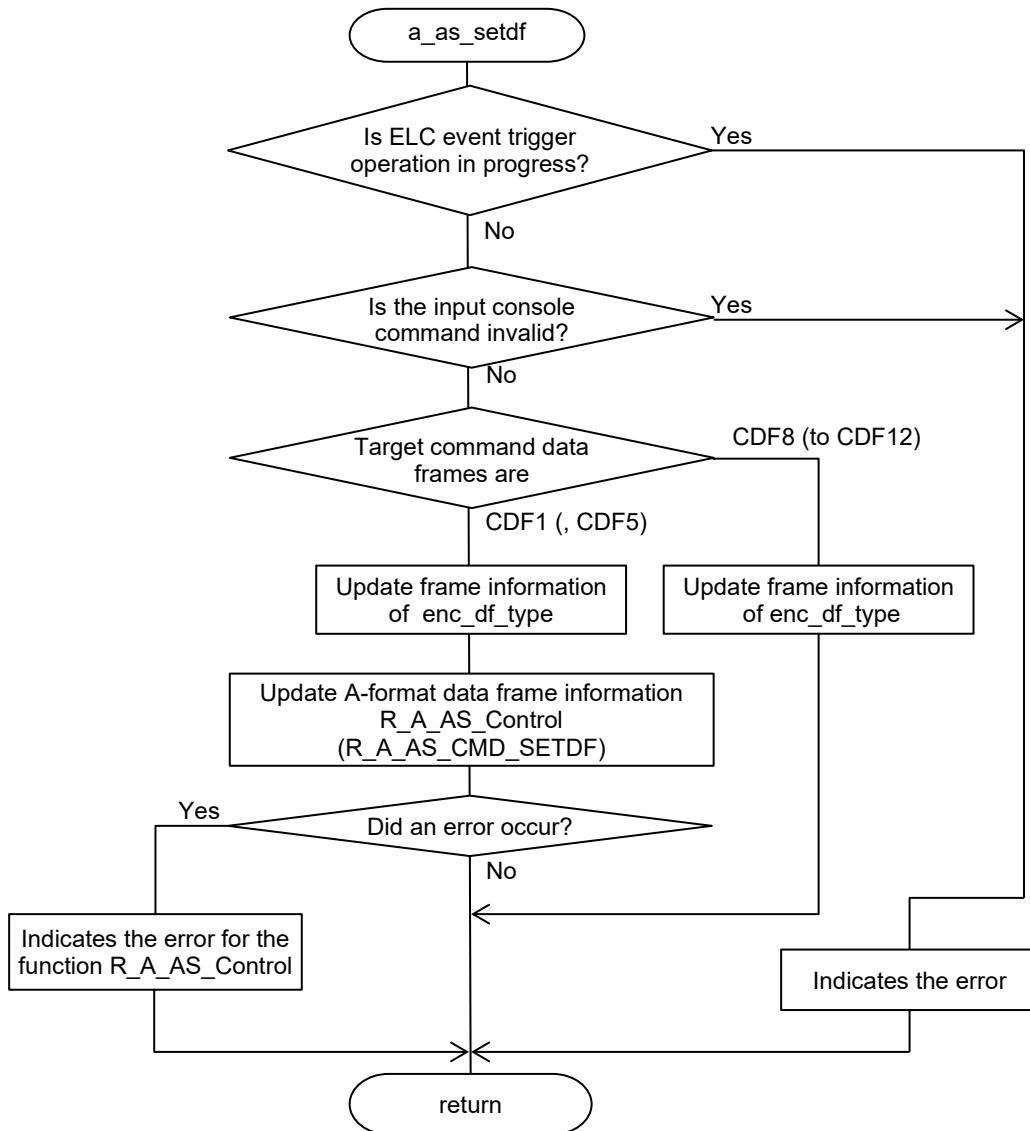


Figure 4.6 Flowchart of a_as_setdf Function

(5) Flowchart of a_as_elctimer

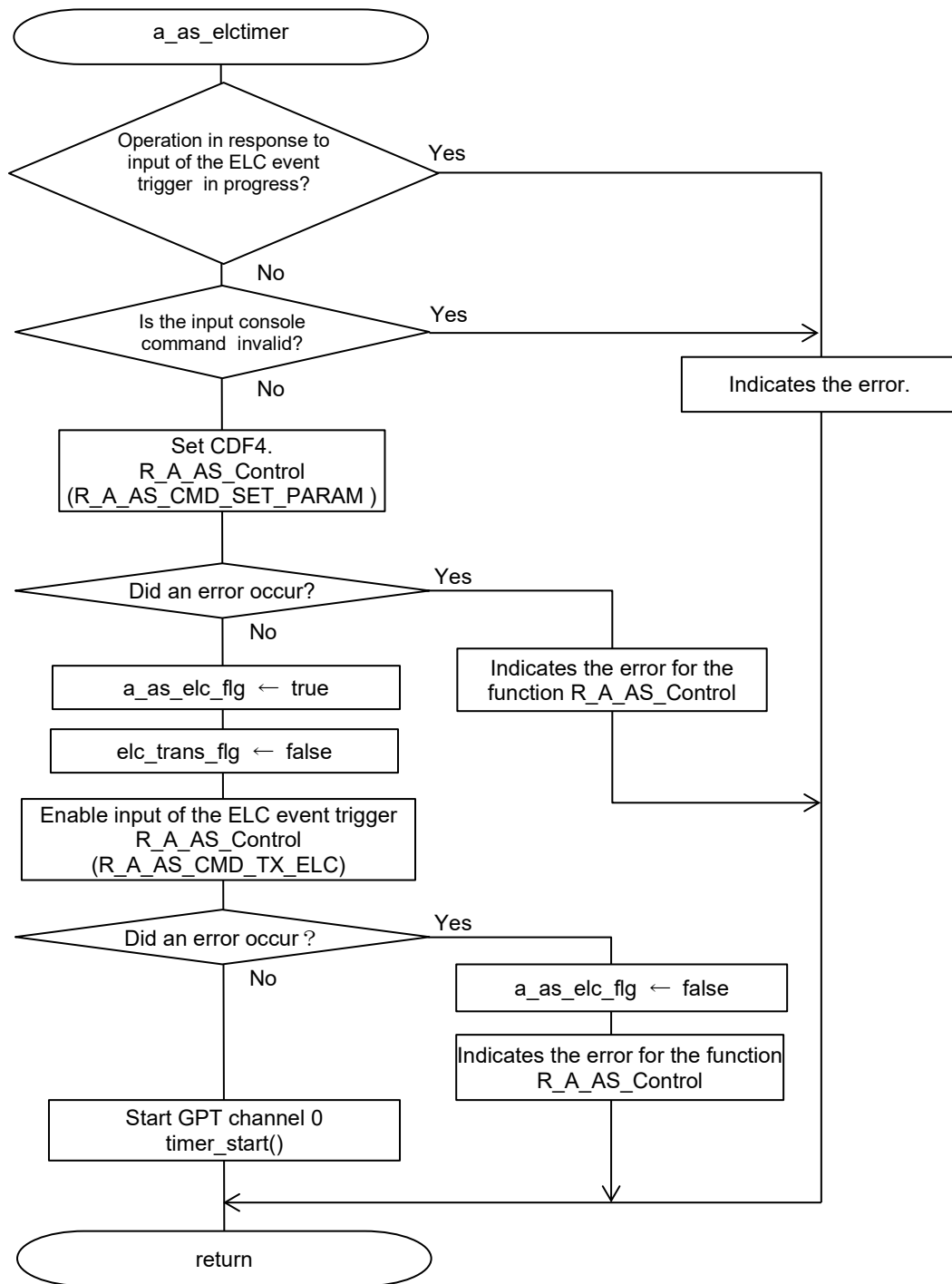


Figure 4.7 Flowchart of a_as_elctimer Function

(6) Flowchart of a_as_elcstop

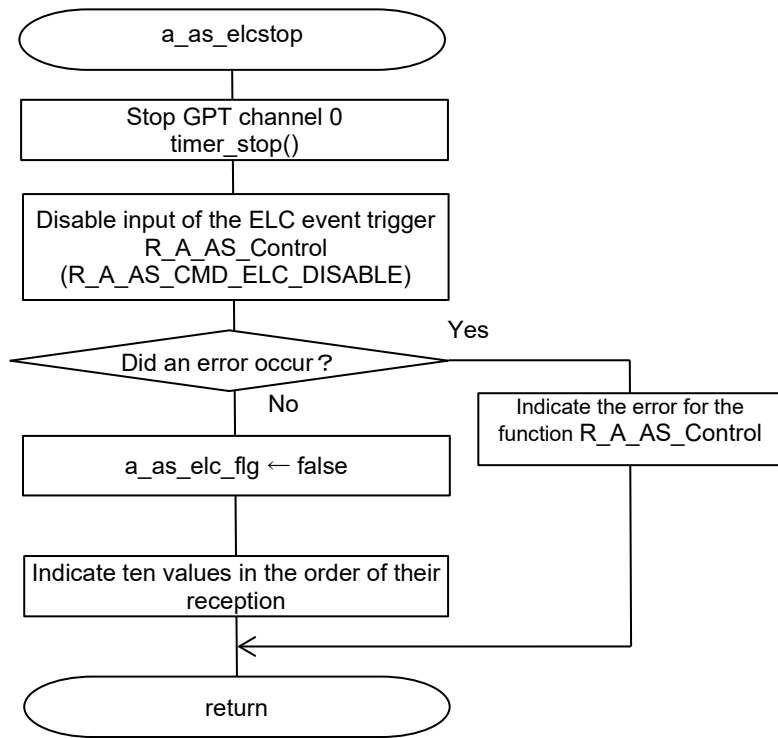


Figure 4.8 Flowchart of a_as_elcstop Function

(7) Flowchart a_as_txerr_callback

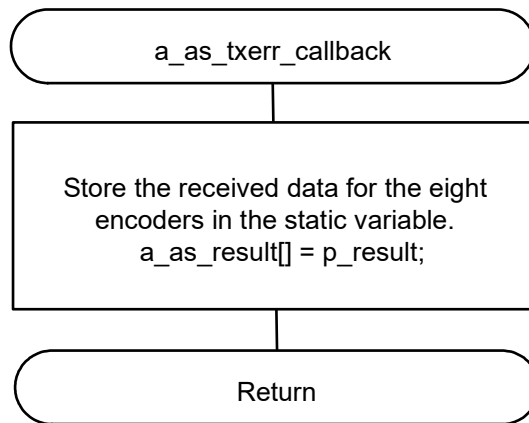


Figure 4.9 Flowchart of a_as_txerr_callback Function

(8) Flowchart of a_as_rxset_callback

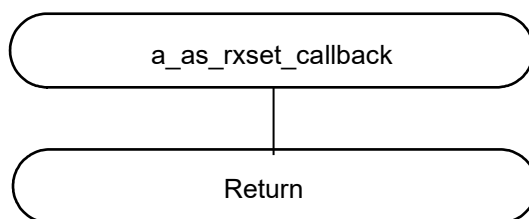


Figure 4.10 Flowchart of a_as_rxset_callback Function

(9) Flowchart of a_as_rxend_callback

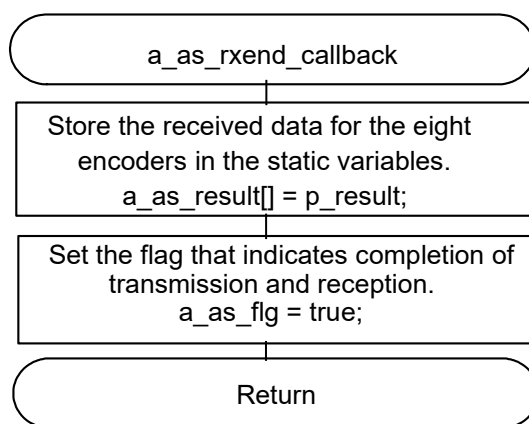


Figure 4.11 Flowchart of a_as_rxend_callback Function

(10) Flowchart of a_as_elctimer_callback

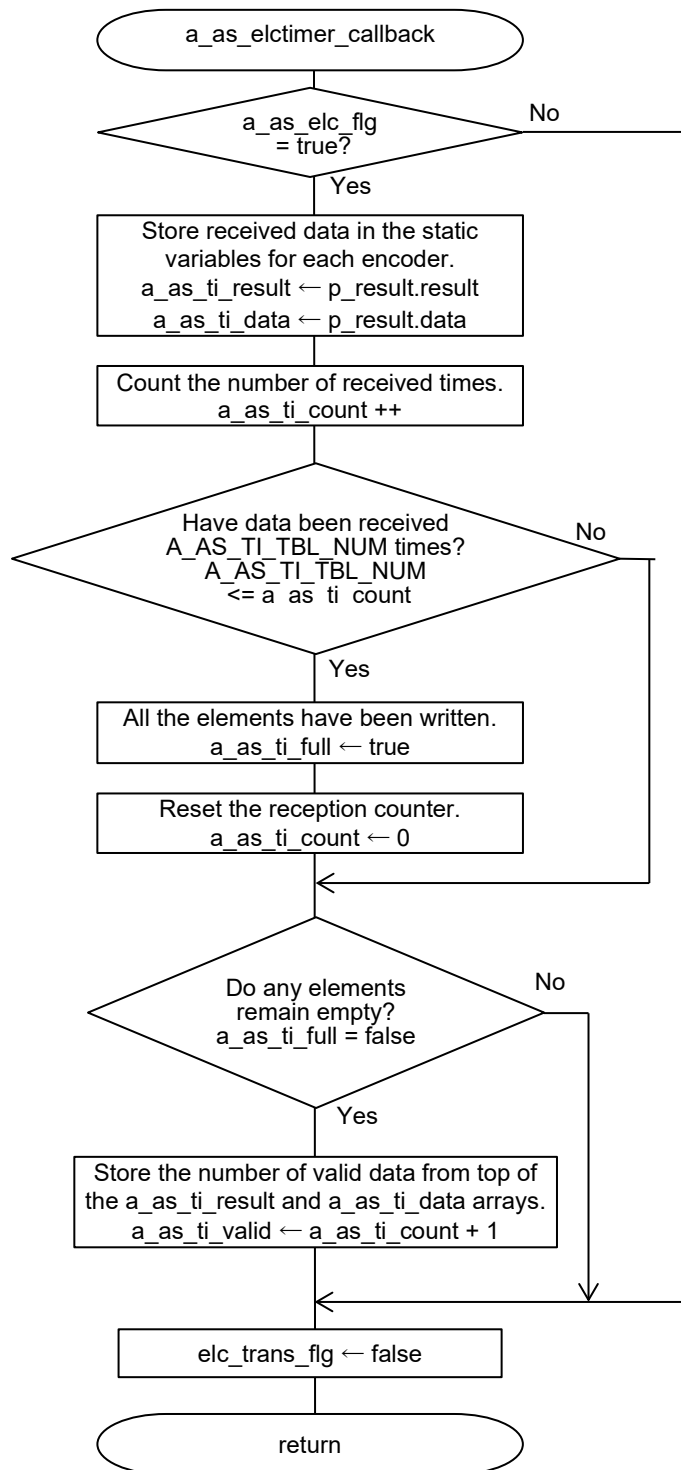


Figure 4.12 Flowchart of a_as_elctimer_callback Function

(11) Flowchart of endat_pos

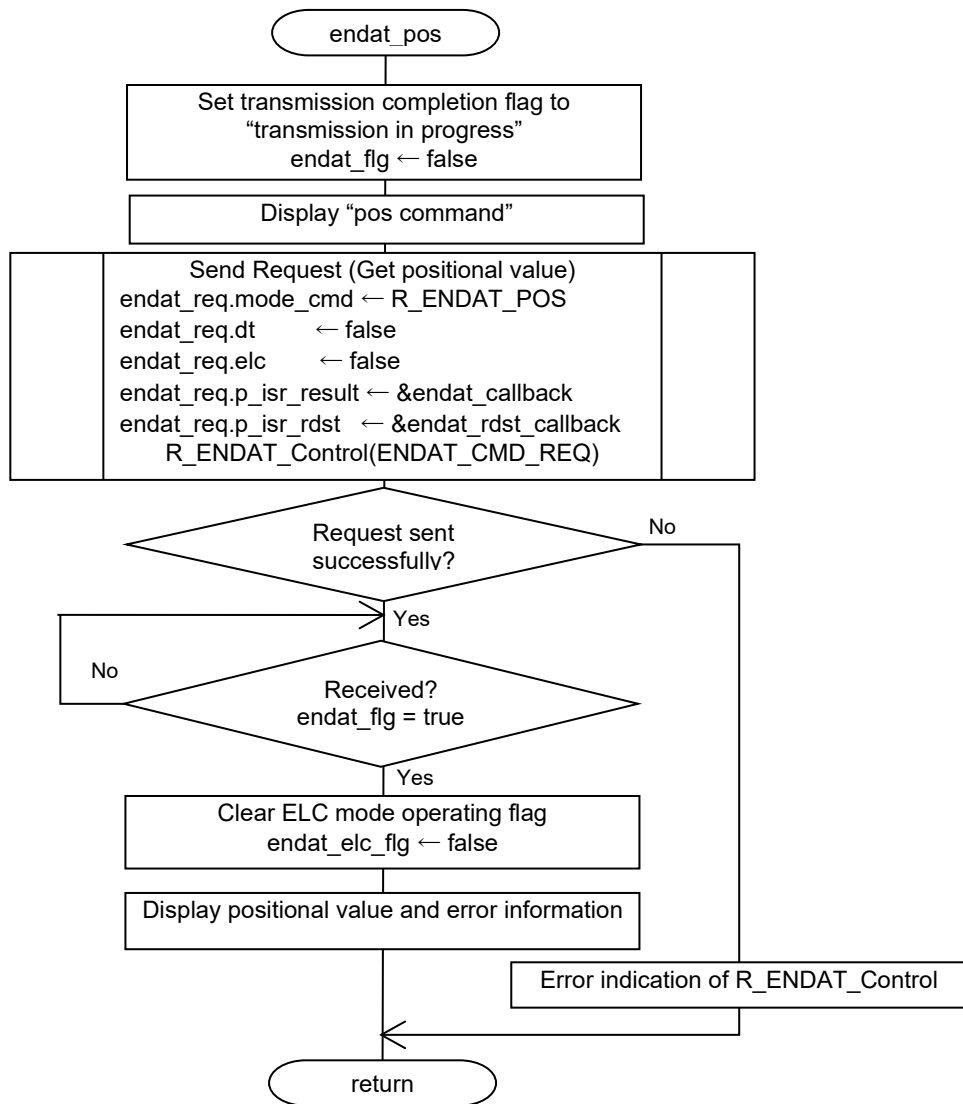


Figure 4.13 Flowchart of endat_pos Function

(12) Flowchart of endat_poscon

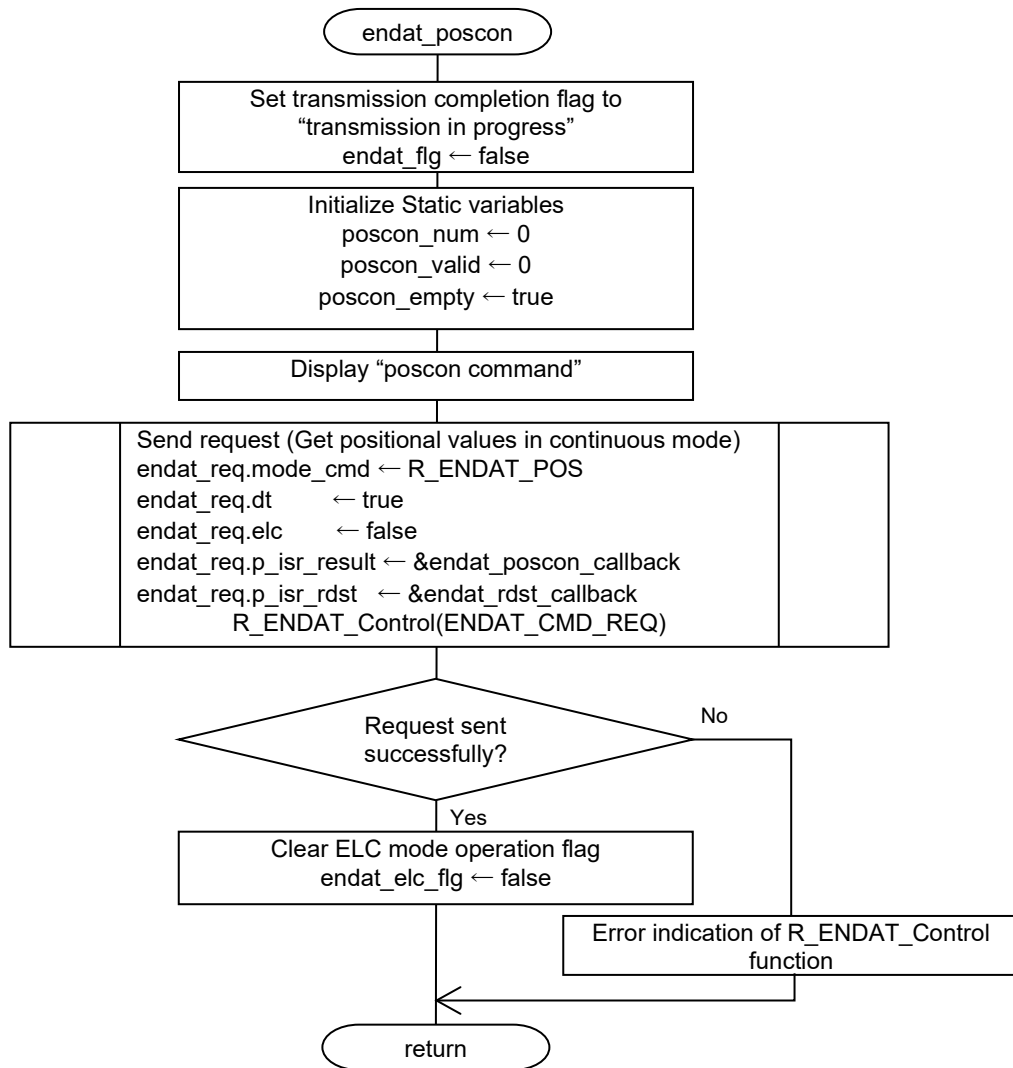


Figure 4.14 Flowchart of endat_poscon Function

(13) Flowchart of endat_elctimer

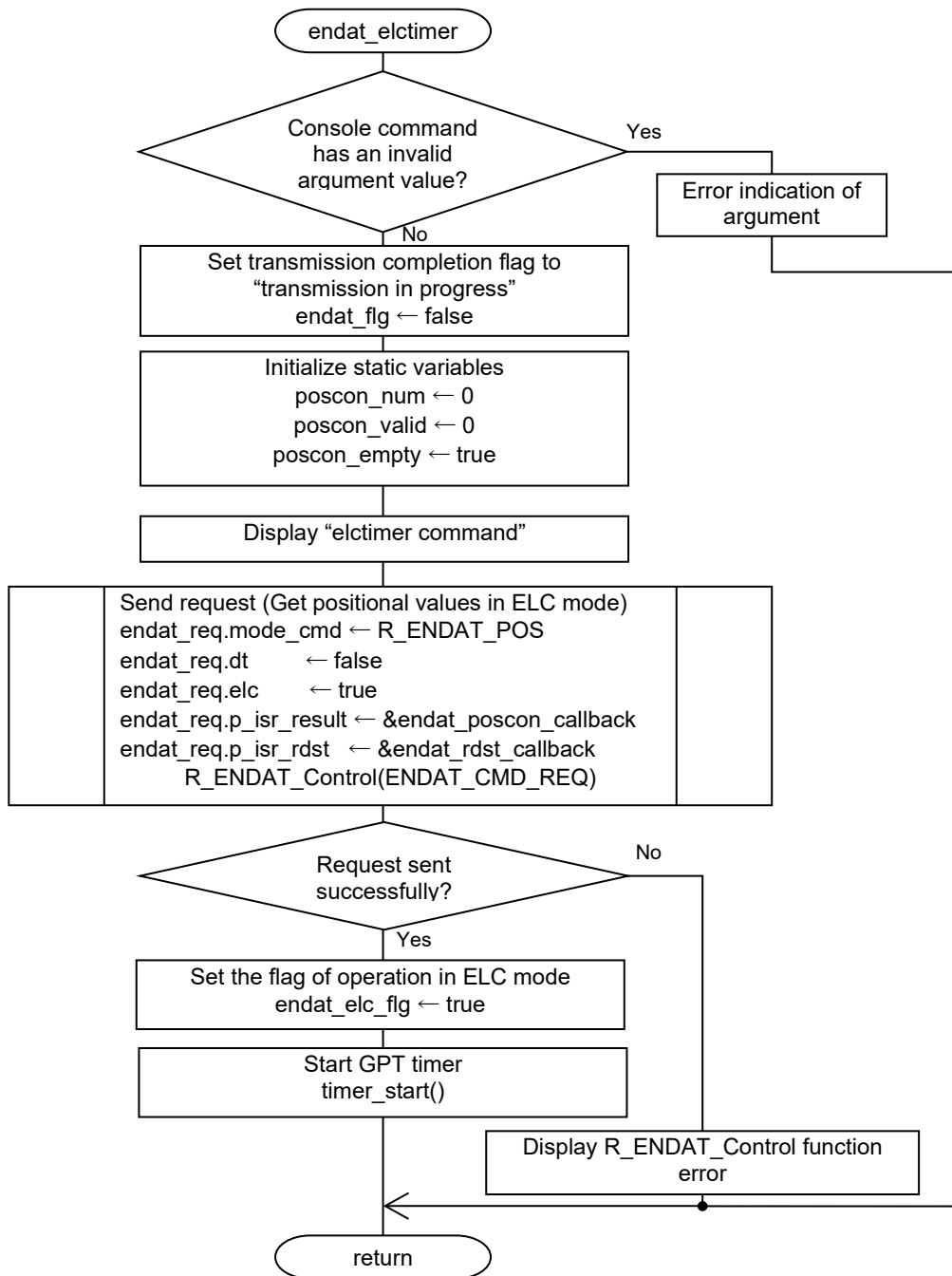


Figure 4.15 Flowchart of endat_elctimer Function

(14) Flowchart of endat_stop

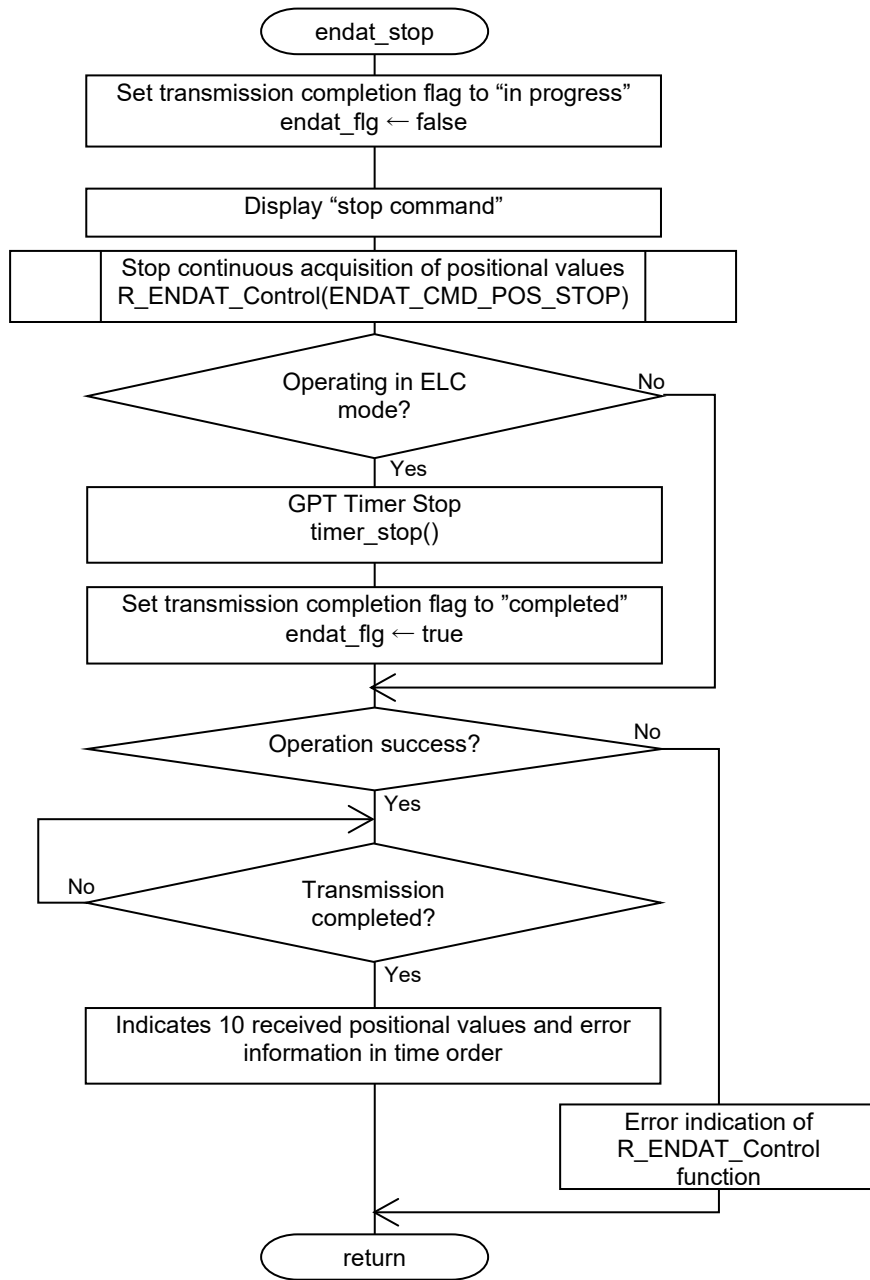


Figure 4.16 Flowchart of endat_stop Function

(15) Flowchart of endat_temp

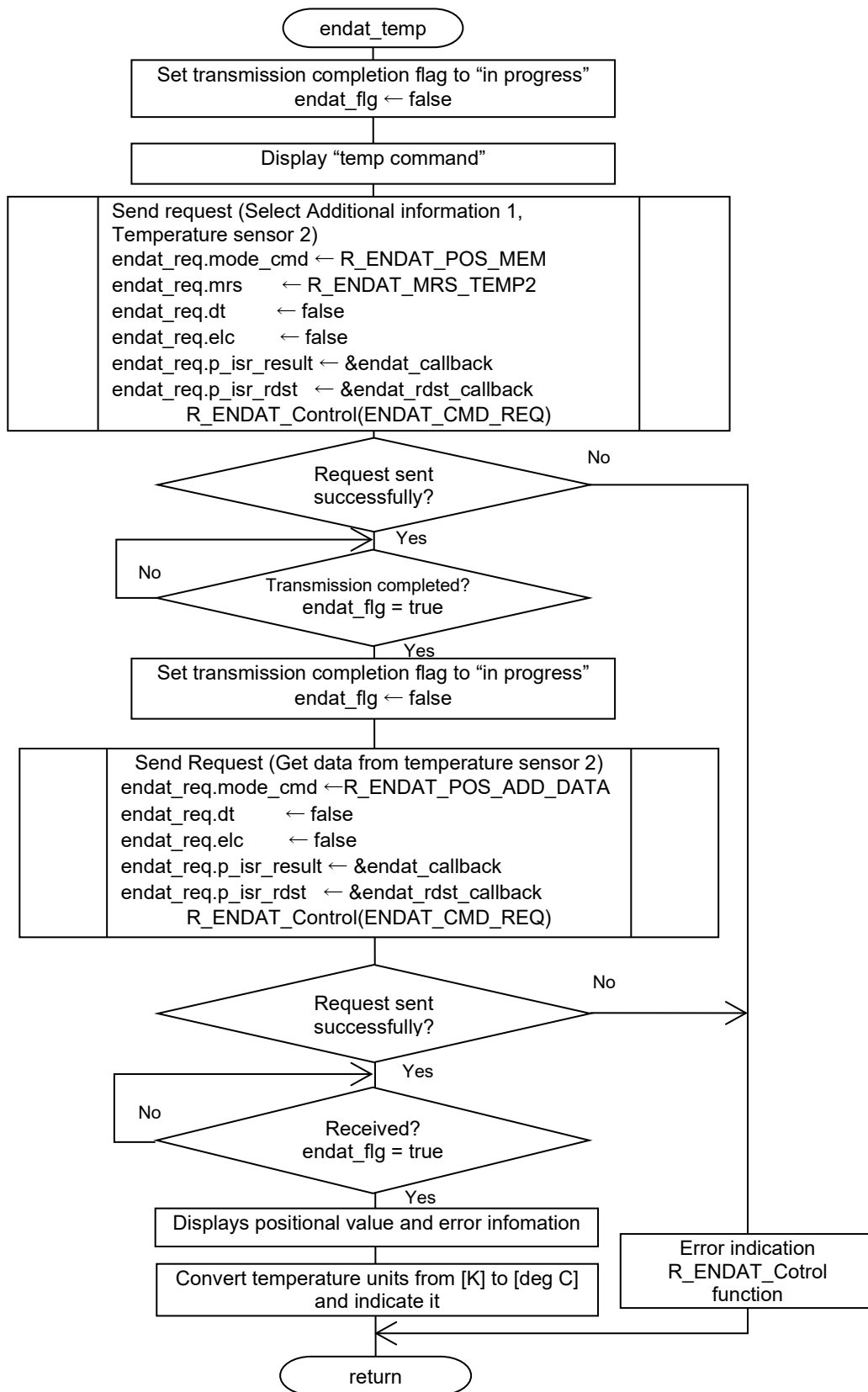


Figure 4.17 Flowchart of endat_temp Function

(16) Flowchart of endat_callback

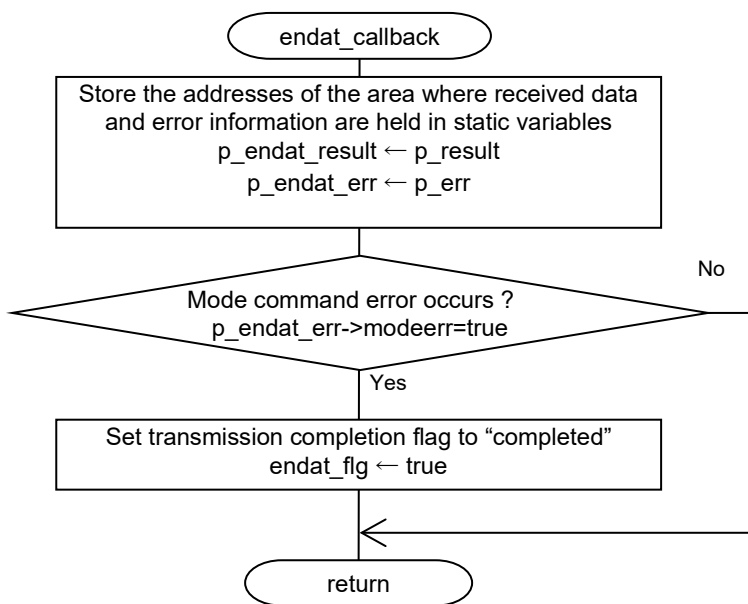


Figure 4.18 Flowchart of endat_callback Function

(17) Flowchart of endat_poscon_callback

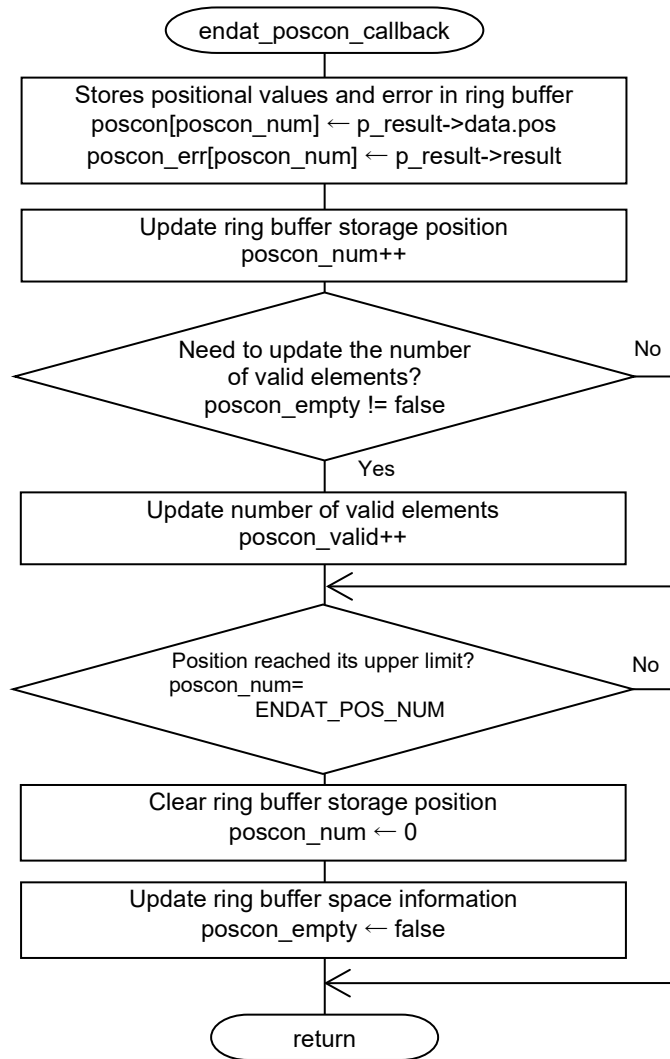


Figure 4.19 Flowchart of endat_poscon_callback Function

(18) Flowchart of endat_rdst_callback

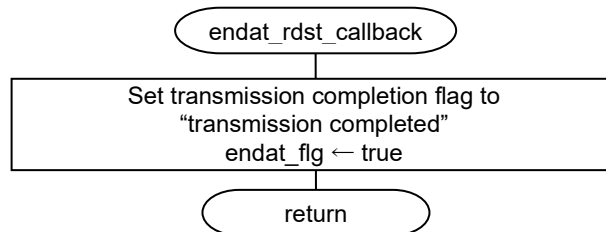


Figure 4.20 Flowchart of endat_rdst_callback Function

4.15.9 Dual Encoder Operation Sequence

(1) Dual Encoder Startup Sequence

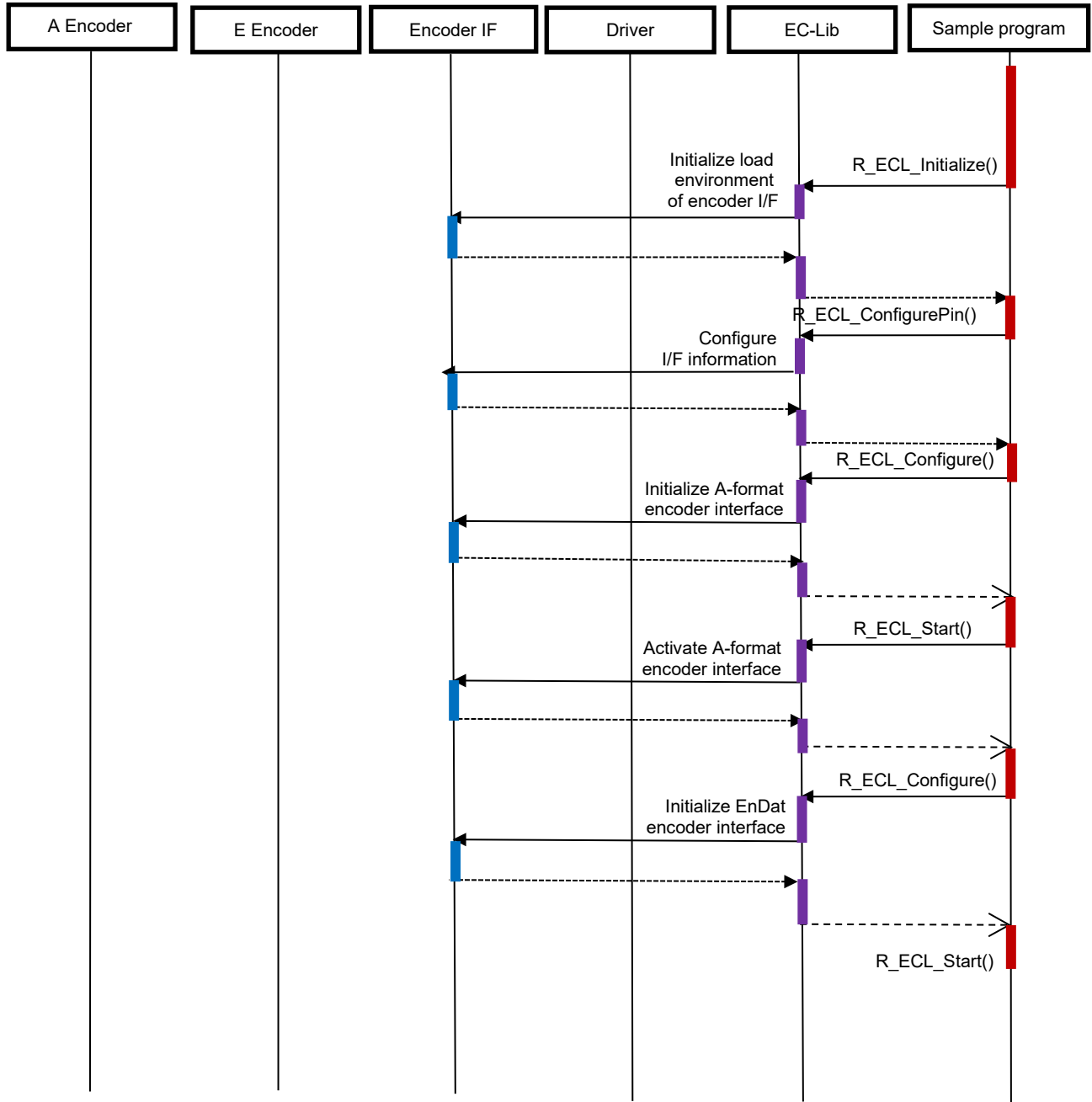


Figure 4.21 Dual Encoder Startup Sequence Diagram (1/2)

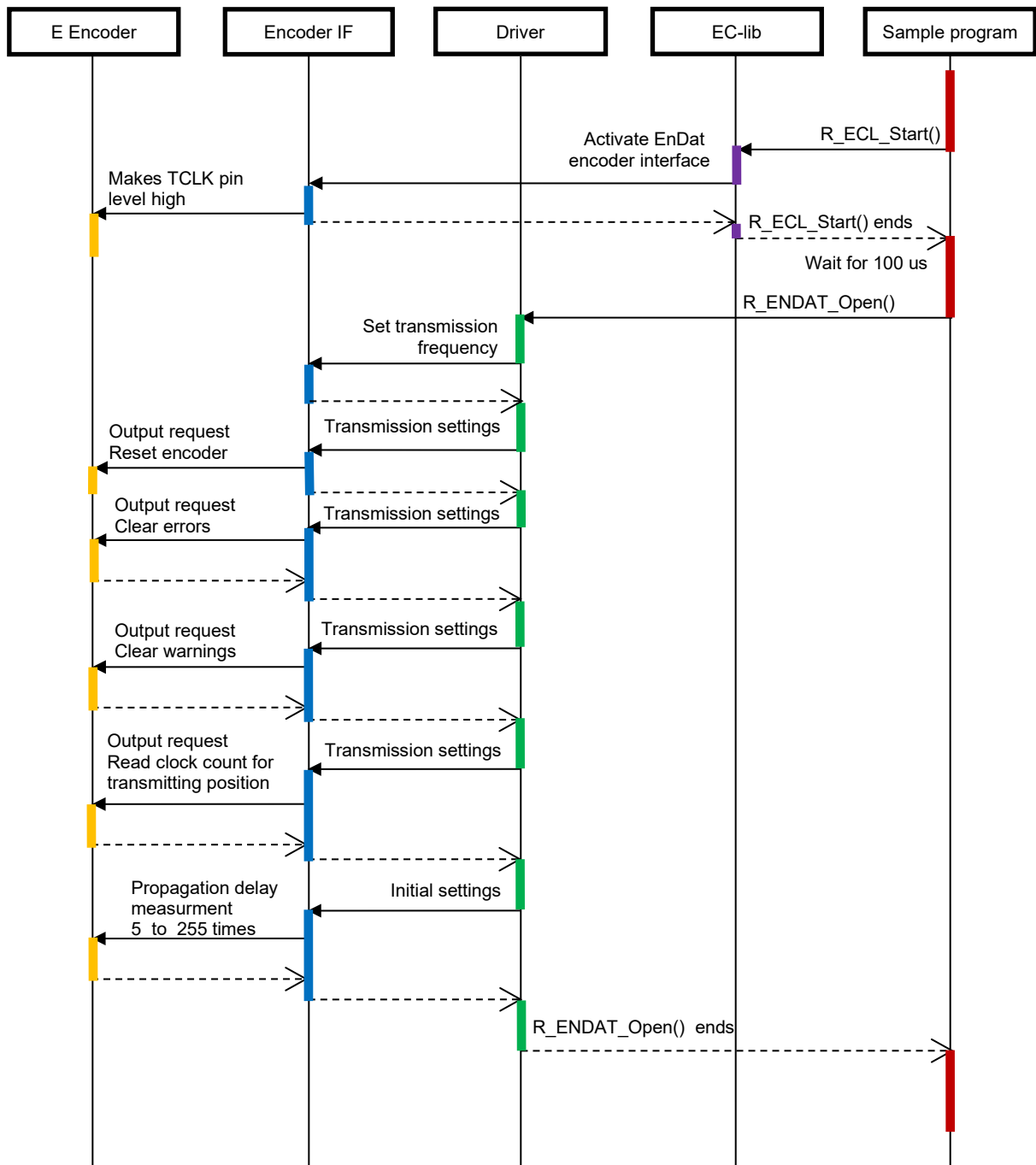


Figure 4.22 Dual Encoder Startup Sequence Diagram (2/2)

(2) A-format Request Transmission and Data Reception Sequence

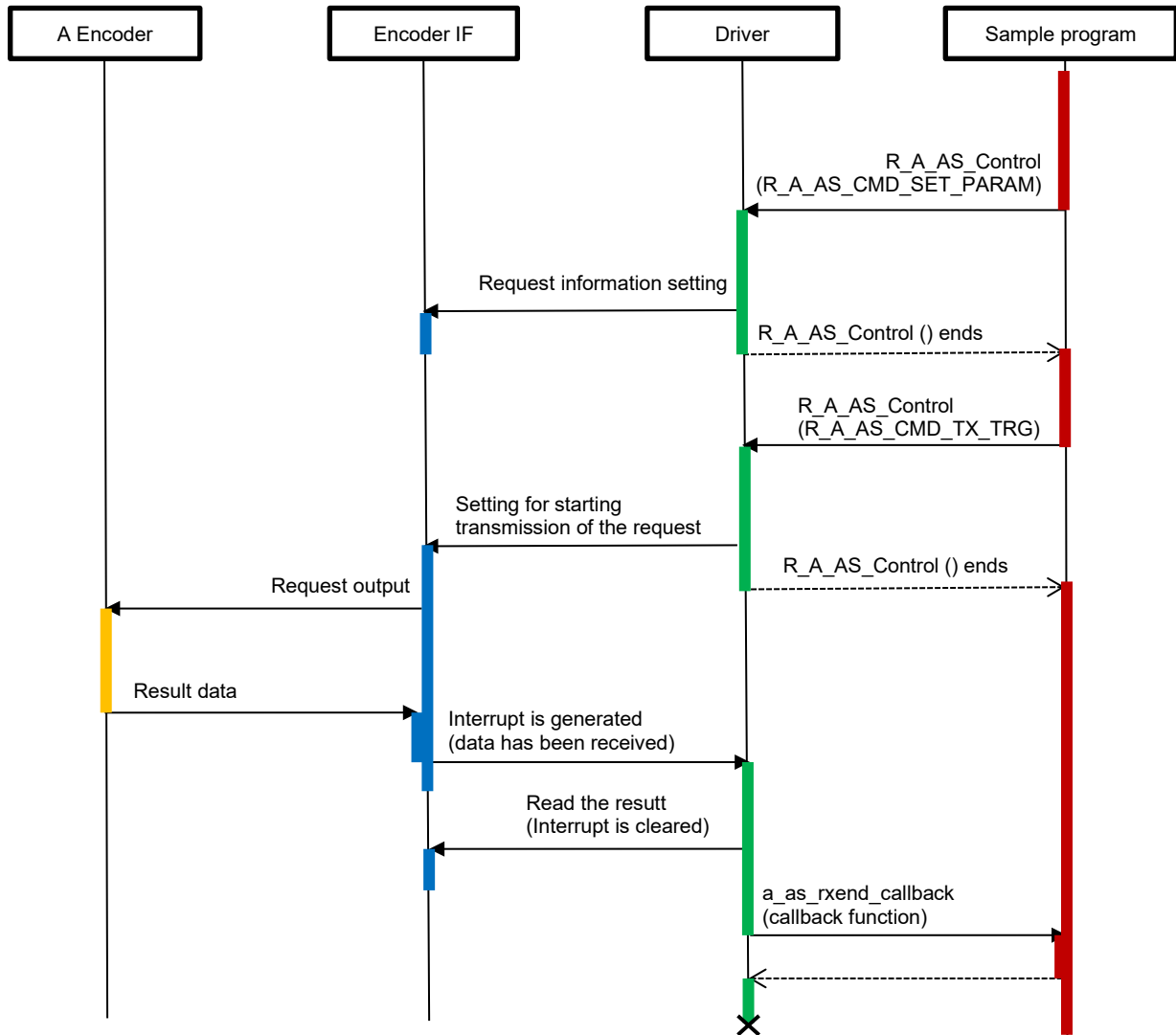


Figure 4.23 A-format Request Transmission and Data Reception Sequence Diagram

(3) EnDat Request Transmission and Data Reception Sequence

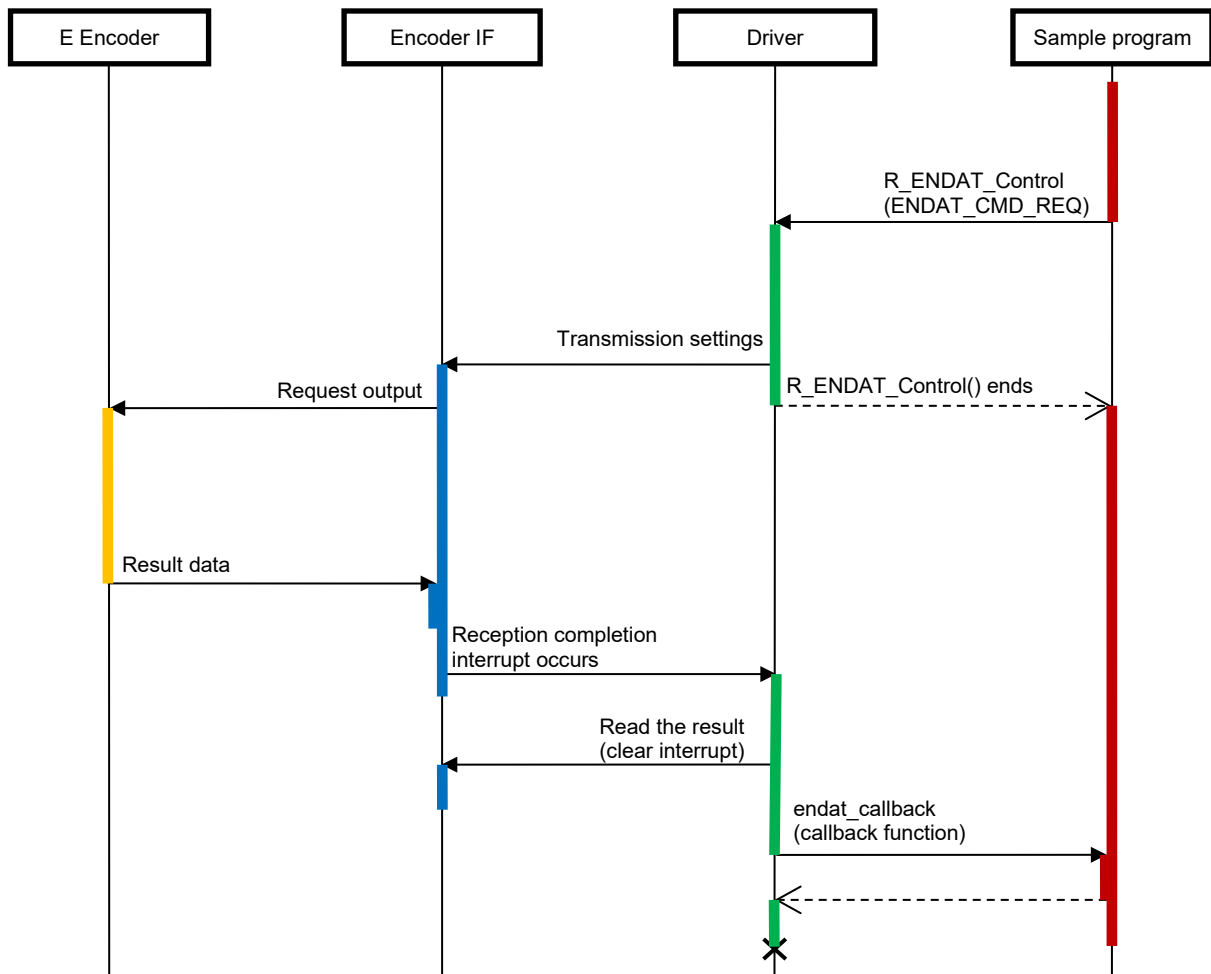


Figure 4.24 EnDat Request Transmission and Data Reception Sequence Diagram

(4) EnDat Request Transmission (Continuous Mode) and Data Reception Sequence

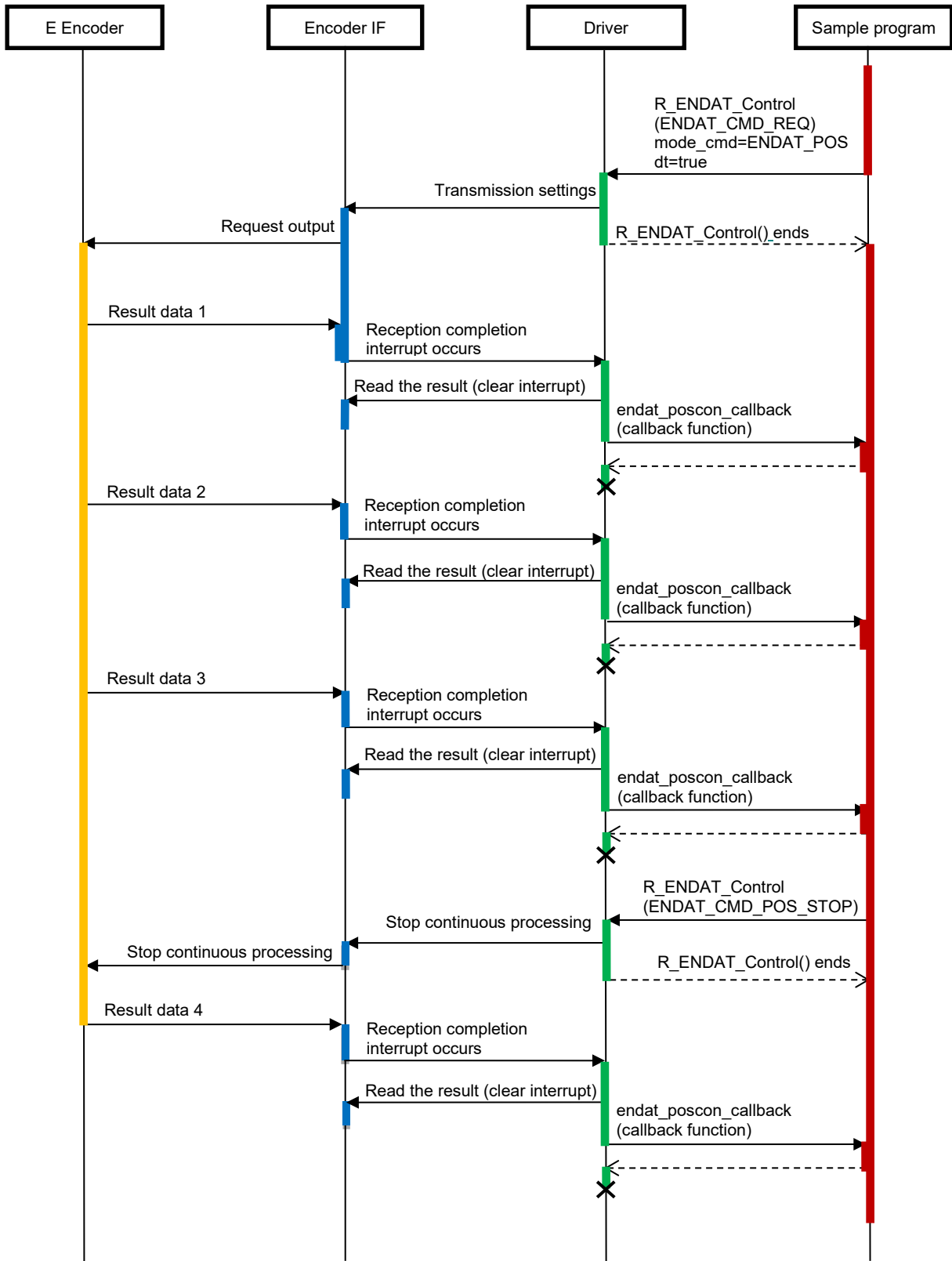


Figure 4.25 EnDat Request Transmission (Continuous Mode) and Data Reception Sequence Diagram

(5) EnDat Request Transmission (ELC Mode) and Data Reception Sequence

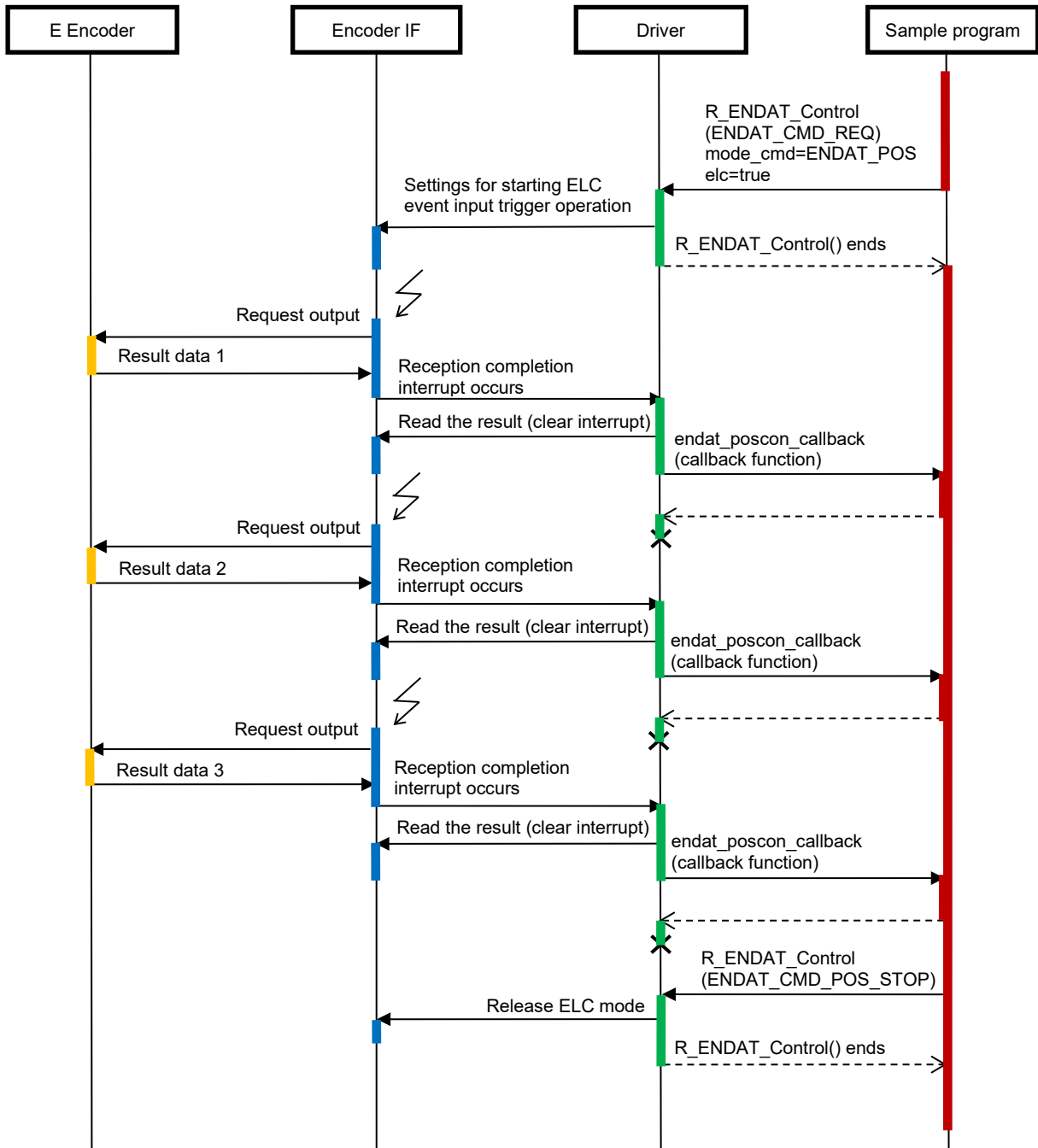


Figure 4.26 EnDat Request Transmission (ELC Mode) and Data Reception Sequence Diagram

(6) A-format ELC Event Trigger Operation Sequence

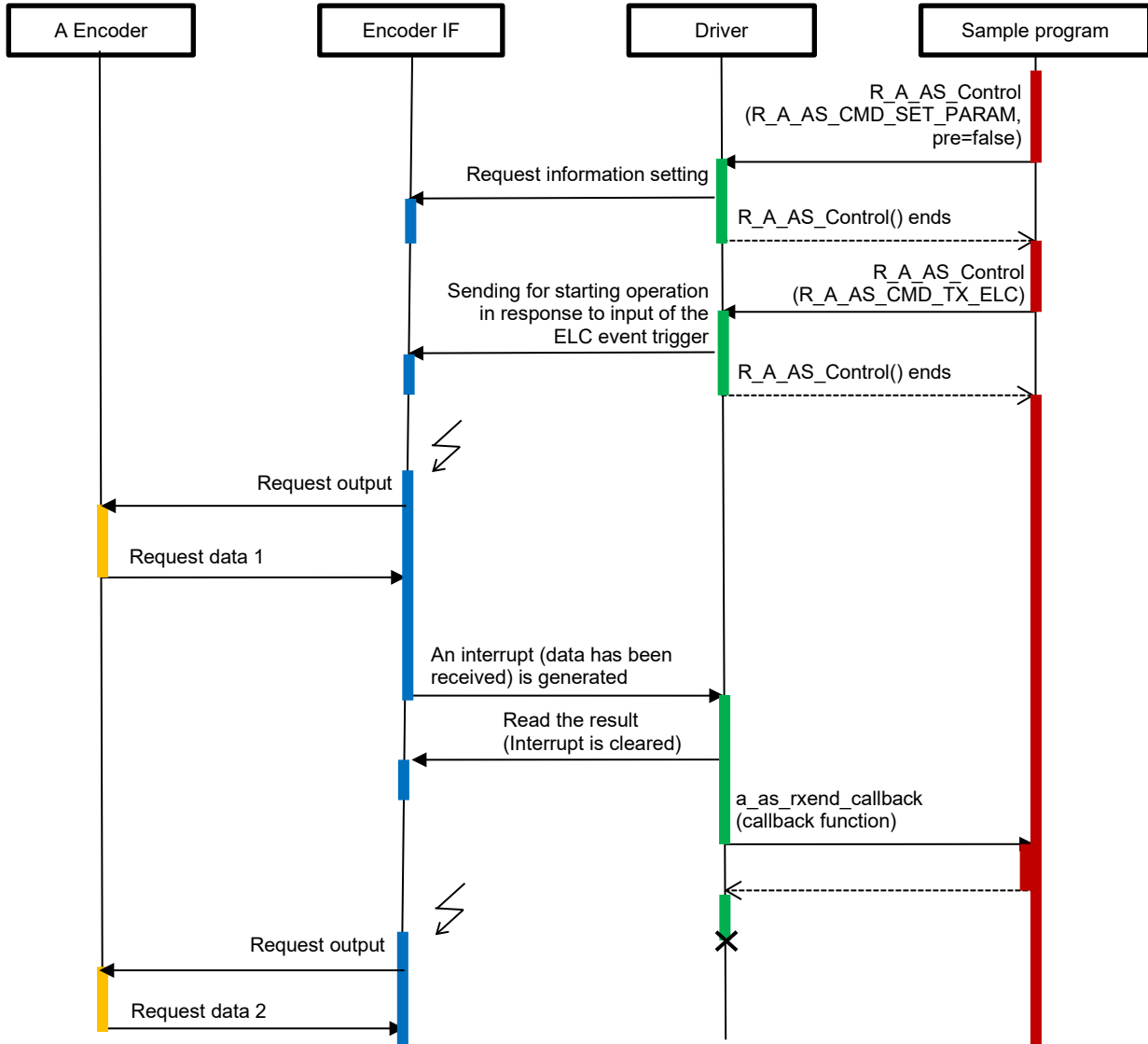


Figure 4.27 A-format ELC Event Trigger Operation Sequence Diagram (1/2)

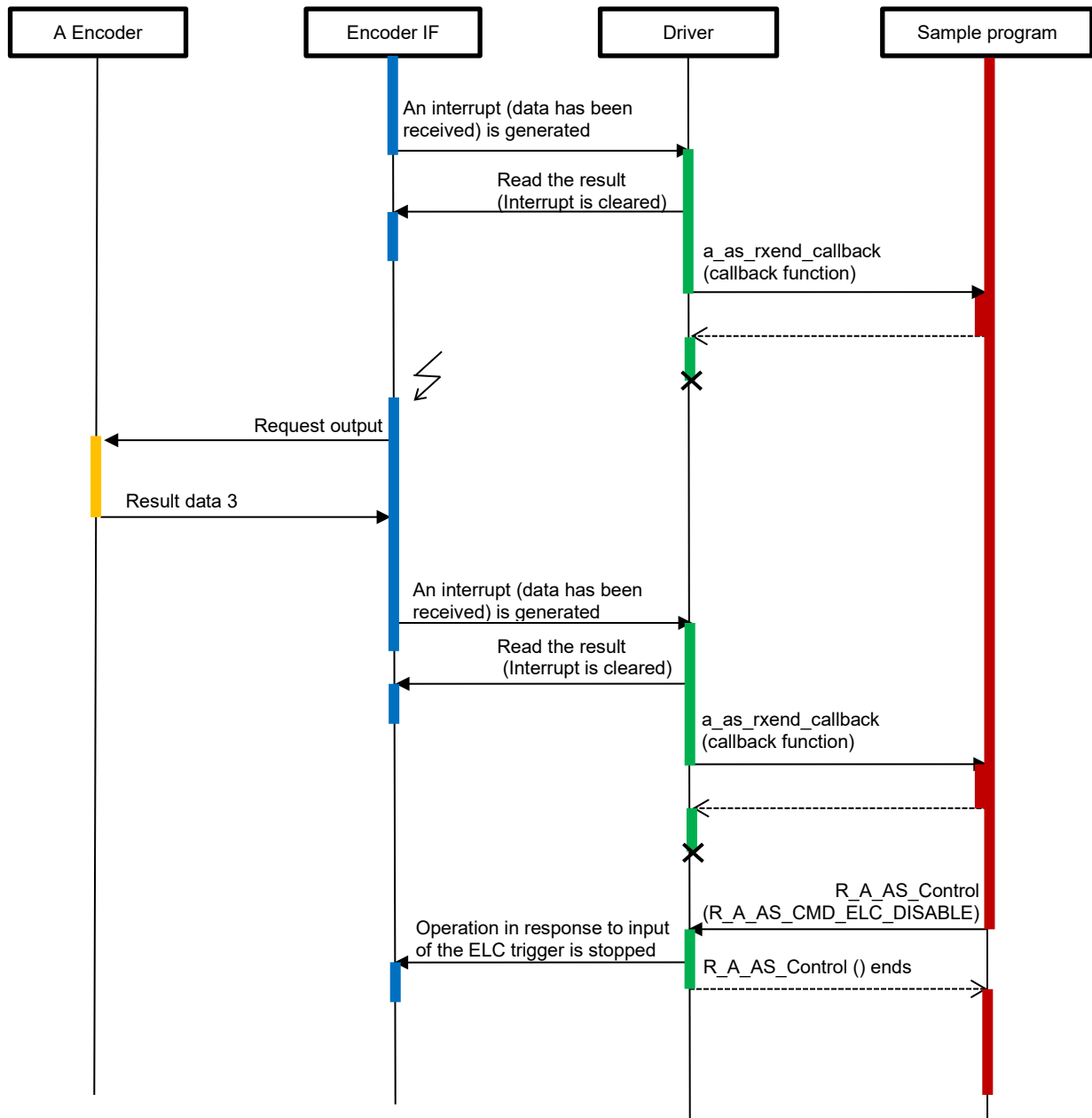


Figure 4.28 A-format ELC Event Trigger Operation Sequence Diagram (2/2)

(7) A-format Stop Sequence

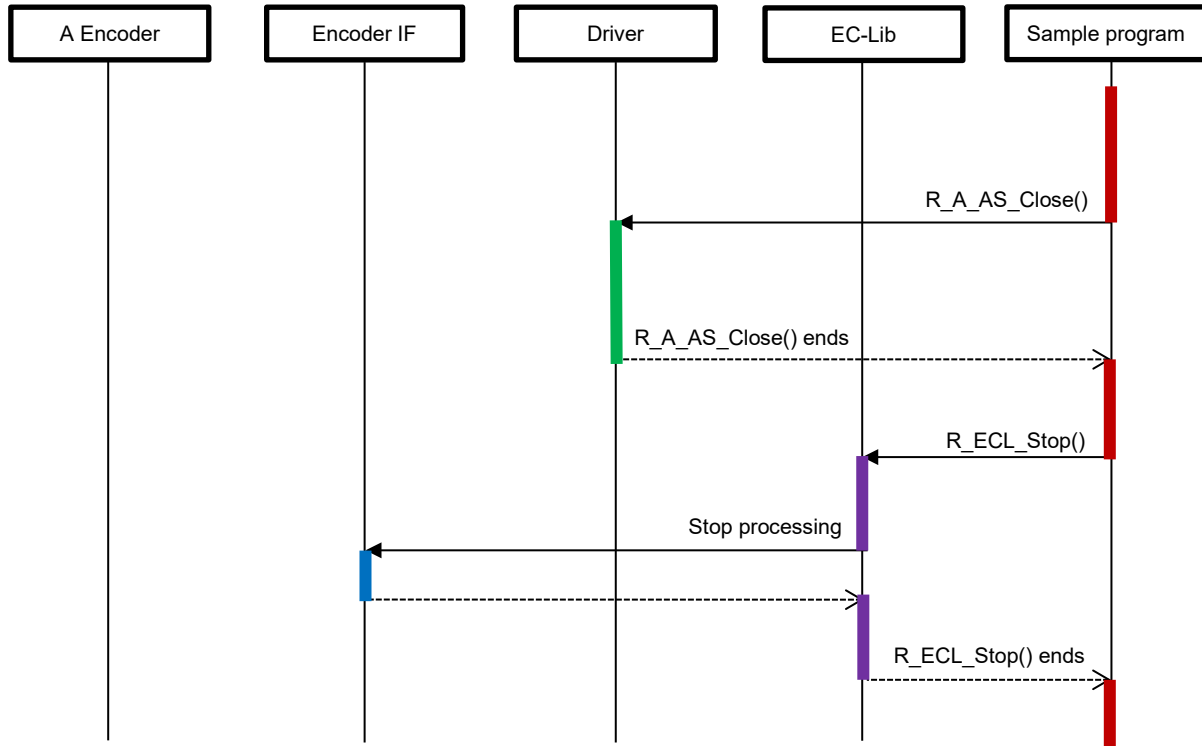


Figure 4.29 A-format Stop Sequence Diagram

(8) EnDat Stop Sequence

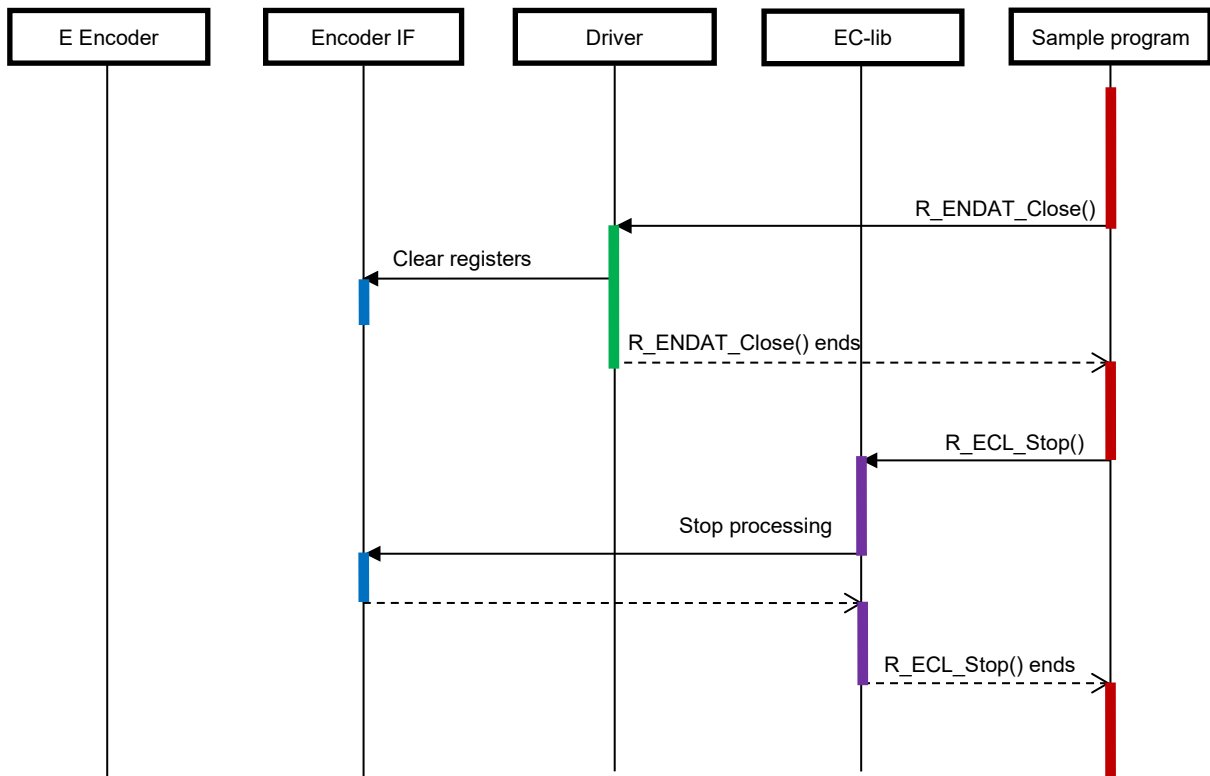


Figure 4.30 EnDat Stop Sequence Diagram

4.15.10 Console Commands

This sample program supports an encoder (MAR-M50A or SAR-HL700A) compliant with the A-format specification and an encoder (EQN1035) compliant with the EnDat 2.2 specification. The commands that can be input from the console are as follows. Commands are prefixed with the identification code "A" or "E" to distinguish the target encoder of the command.

Table 4.30 A-format Console Commands

Command	Content
A req <i>ea cmd param1 param2</i>	Sends the command frame input in the <i>cmd</i> to the encoder. <i>ea</i> : Encoder section number (decimal notation) <i>cmd</i> : Command frame <i>param1</i> and <i>param2</i> : The values to be input depends on the <i>cmd</i> value. See "Table 4.33, A-format req Commands and Corresponding Parameters".
setdf <i>ea cmd encmod</i>	Sets data frame types of the encoder. * <i>ea</i> : Encoder section number (decimal notation) <i>cmd</i> : Target command data frame (CDF1: Data frame type of CDF1 and CDF5, CDF8: Data frame type of CDF8 to CDF12) <i>encmod</i> : Extended data frame is used or not (0: Data frame is not extended, Others: Data frame is extended. ABS full 40-bit + velocity is selected for CDF1 and CDF5. ABS LSB 24-bit is selected for CDF8 to CDF12.) Usage example (For the section number 2 encoder, data frame types of CDF1 and CDF5 are treated as ABS full 40-bit + velocity) A setdf 2 CDF1 1
A elctimer <i>val</i>	Sends the command frame CDF4 for eight encoders at 4 Mbps by the ELC event trigger. <i>val</i> : Timing of the trigger in microseconds. The maximum value allowed is 6990. Operation in response to input of the ELC event trigger is stopped by executing the console command "elcstop".
A elcstop	Stops the operation in response to input of the ELC event trigger.

Note: 1. This command is used for A-format version 3.0 encoders or later. Data frame types for the CDF1, CDF5 and CDF8 to CDF12 are predetermined at the shipping of the encoders. Set data frame types to suit each encoder.

Table 4.31 EnDat Console Commands

Command	Content
E pos	Get positional value only once.
E poscon	The positional values are acquired continuously. To stop continuous acquisition, enter the "stop" command.
E elctimer val	The positional value is continuously acquired in a timer cycle as an ELC event input trigger operation. The timer cycle is specified in units of us (maximum 6990us). To stop continuous acquisition, enter the "stop" command.
E stop	Stops continuous acquisition of positional values.
E temp	Obtains temperature measurements along with positional values from the encoder.

Table 4.32 Common Console Commands

command	content
exit	Exit the program.

Table 4.33 A-format req Commands and Corresponding Parameters

cmd	param1	param2
CDF0 to CDF12 *1	None	None
CDF13	An address in EEPROM (hexadecimal notation) Specify a value between 0x00 to 0xFF.	None
	Usage example (reading the data from address 0) A req 2 CDF13 00	
CDF14	An address in EEPROM (in hexadecimal) Specify a value between 0x00 to 0xEF	The data to be written to EEPROM (in hexadecimal) Specify a value between 0x0000 to 0xFFFF.
	Usage example: writing 0x1234 to address 0 of the encoder with section number 2. A req 2 CDF14 00 1234	
CDF15 to CDF17	None	None
CDF18 to CDF20	Recognition code (in hexadecimal)	None
	Usage example: writing 0x12 3456 to the recognition code of the encoder with section number 2. A req 2 CDF18 123456	
CDF21, CDF22, CDF27 to CDF30	None	None
CDF23 to CDF26 *2	None	None
CDF13x *2	A Bank and an address of EEPROM (in hexadecimal) Specify a value between 0x0000 to 0xFFFF. MSB 8 bits of the 16 bits are used as the bank number. LSB 8 bits are an address.	None
	Usage example (reading the data from bank 1, address 0 of the section 2 encoder) A req 2 CDF13x 100	
CDF14x *2	A Bank and an address of EEPROM (in hexadecimal) Specify a value between 0x0000 to 0xFFFF. MSB 8 bits of the 16 bits are used as the bank number. LSB 8 bits are an address.	The data to be written to EEPROM (in hexadecimal) Specify a value between 0x0000 to 0xFFFF.
	Usage example (writing 0x1234 to bank 1, address 0 of the section 2 encoder) A req 2 CDF14x 100 1234	
CDF16x *2	None	None
CDF18x *2	Velocity coefficient (in hexadecimal) Specify a value between 0x0 0000 to 0x7 FFFF.	None
	Usage example (writing 0x1 2345 to velocity coefficient of the section 2 encoder) req 2 CDF18x 12345	

Note: 1. Though input of CDF11, CDF17, and CDF19 are possible, they cannot be used because the sample program is for operation with a bus connection.
2. This is a command data frame available for A-format version 3.0 encoders or later.

(1) Result of Running

After running, it will display the command prompt following the version. Enter the command after "Dual >".

```
Dual encoder sample program start
R_A_AS_GetVersion = 4.0

R_ENDAT_GetVersion = 4.0

Dual>
```

(2) Example of Command Execution

It is an example of executing console command that sends command frame CDF0 to a connecting encoder ENC1. Encoder address, status and angular information are displayed as the response from the encoder.

```
Dual>A req 1 CDF0
A req command
A -----
A ENC1
A R_A_AS_REQ_SUCCESS
A EA : 0
A ES : 0
A CC : 0
A ABS 40bit [39:32] : 0x00000005
A ABS 40bit [31:0] : 0x00560E11
```

It is an example of executing E pos command. Results of sending and receiving requests, received positional value and additional data are displayed as the response from the encoder.

```
Dual>E pos
E pos command
E result      : ENDAT_SUCCESS
E pos_upper   : 0x00000000
E pos_lower   : 0x00778D1B
E add_datum1  : 0x00000000
E add_datum2  : 0x00000000
```

4.15.11 Procedure of Channel Switching

To switch EnDat with channel 0 and A-format with channel 1, the macros defined in the source code dual_main.c and the interrupt handler specified in the FSP Configurator must be changed.

Table 4.34 Changes in Macro Definitions (dual_main.c)

Original macro definition	Macro definitions for channel switching
<code>#define ch0_id "A"</code>	<code>#define ch0_id "E"</code>
<code>#define ch1_id "E"</code>	<code>#define ch1_id "A"</code>
<code>#define A_AS_CH (0)</code>	<code>#define A_AS_CH (1)</code>
<code>#define ENDAT_CH (1)</code>	<code>#define ENDAT_CH (0)</code>

In the Interrupts Configuration tab of the FSP Smart Configurator configuration screen, change the interrupt handler function (ISR) assignment.

Table 4.35 Original interrupt handler

Interrupt	Event	ISR
372	ENCIF_INT0 (ENCIF CH0 Interrupt A)	a_as0_int_isr
376	ENCIF_INT4 (ENCIF CH1 Interrupt A)	endat1_rx_int_isr

Table 4.36 Interrupt handler for channel switching

Interrupt	Event	ISR
372	ENCIF_INT0 (ENCIF CH0 Interrupt A)	endat0_rx_int_isr
376	ENCIF_INT4 (ENCIF CH1 Interrupt A)	a_as1_int_isr

5. Sample Code

The sample code is available from the Renesas Electronics website.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Nov.30.22		First Edition issued
2.00	Jun 28.24	5 6 14 to 16, 20 26 27 33 34 to 36 40 42 49 50 54, 57, 58, 69 73 to 81	Update description of the board name. Change description order of used pins. Correct description of the reference. Add description of channel switching case. Correct explanation of the tables. Remove description about location of integer type definition. Correct description of the reference. Revise the table description. Correct description of the structure e_endat_result_t. Correct description of the timer_start, timer_stop function arguments. Correct description of the reference. Correct number of indicated samples by endat_pos function. Revise sequence diagrams.
3.00	Oct 25.25	1, 4, 5 1, 13, 29, 30, 36 to 38, 46, 85 9 to 25 9 to 25 38 48, 51, 63, 84 48, 53 86	Change description for trademarks. Update functions, encoder names and command parameters for supporting A-format V3 specification. Change description of headers for functions. Revise to unify description for functions. Revise description of structure r_a_as_status_t. Add description of a_as_setdf function and setdf command. Add description of a_as_elctimer_callback in user-defined functions Add example of command execution.
4.00	Apr 10.26	9 to 24, 40, 49 to 56 14 to 17, 22 to 25 18, 25, 26, 87	Change prefix of pointer variables to "p_". Revise header column of the specifications of user-defined functions. Change interrupt handler name.

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.

- A-format is a trademark of Nikon Corporation.
- EnDat is a registered trademark of Dr. Johannes Heidenhain GmbH.
- IAR Embedded Workbench is a registered trademark of IAR Systems.
- Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.
- Additionally all product names and service names in this document are a trademark or a registered trademark which belongs to the respective owners.

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/.