RE01 1500KB, 256KB Group

CMSIS Driver R_USART Specifications

Summary

This document describes the detailed specifications of the USART driver provided in the RE01 1500KB and 256KB Group CMSIS software package (hereinafter called the USART driver).

Target Device

RE01 1500KB Group

RE01 256KB Group

Contents

1. Overview

This is a USART driver for RE01 1500KB and 256KB Group devices and is compliant with the ARMS's basic CMSIS software standard. This driver uses the following peripheral functions.

Table 1-1 Peripheral functions used by the R_USART driver

Peripheral functions	Description
Serial communication interface (SCI)	Asynchronous and synchronous serial communication is realized using SCI.
Data transfer controller (DTC) (Note)	When DTC control is selected, DTC is used to write data to the transmit data register (TDR) and read data from the receive data register (RDR).
DMA controller (DMAC) (Note)	When DMAC control is selected, DMAC is used to write data to the transmit data register (TDR) and read data from the receive data register (RDR).

Note. Used only when DMAC or DTC is specified for communication control. For details, refer to "2.4 Communication Control and NVIC Interrupt Setting".

2. Driver Configuration

This chapter describes the information required for using this driver.

2.1 File Configuration

This USART driver is conform to the CMSIS driver package specification and consists of seven files: "Driver_USART.h" in the ARM CMSIS file storage directory, "r_usart_cmsis_api.c", "r_usart_cmsis_api.h", "r_usart_cfg.h", "R_Driver_USART.h", "pin.c"and "pin.h" in the vendor-specific file storage directory. The functions of the files are shown in Table 2-1, and the file structure is shown in Figure 2-1.

Table 2-1 Functions of R_USART Driver Files

File Name	Description
Driver_USART.h	CMSIS Driver standard header file.
R_Driver_USART.h	CMSIS Driver extended header file. To use the USART driver, it is necessary to include this file.
r_usart_cmsis_api.c	Driver source file. It provides the entity of the driver function. To use the USART driver, it is necessary to build this file.
r_usart_cmsis_api.h	Driver header file. The macro, type, and prototype declarations to be used in the driver are defined.
r_usart_cfg.h	Configuration definition file. It provides configuration definitions that can be modified by the user.
pin.c	Pin setting file. It provides pin assignment processing for various functions.
pin.h	Pin setting header file.

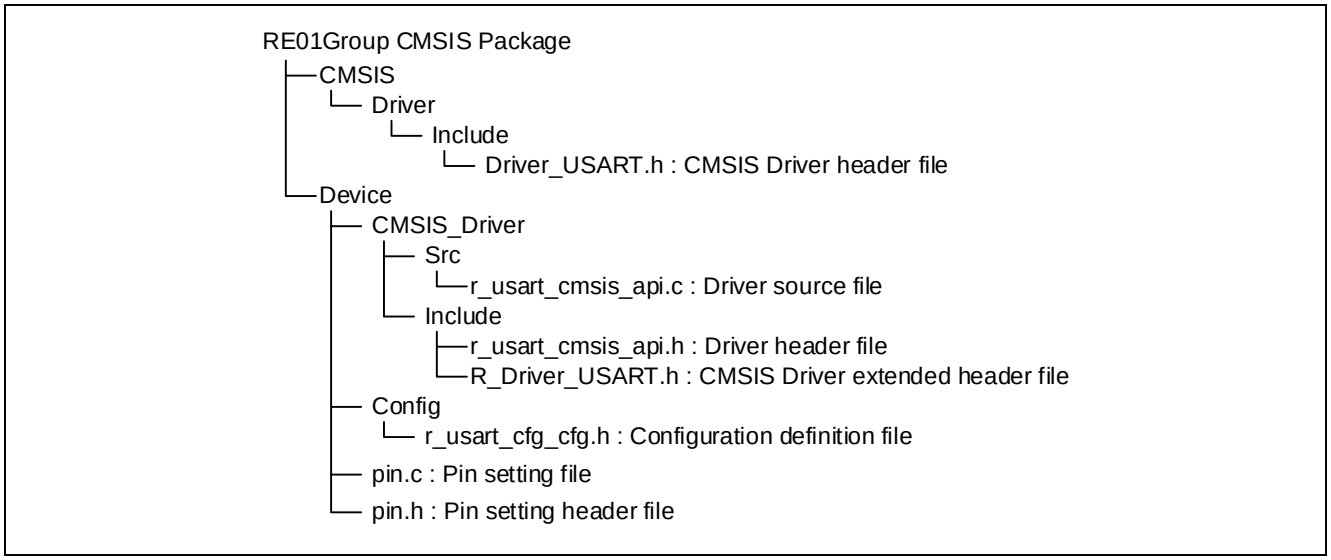


Figure 2-1 File Configuration of USART Driver

2.2 Driver APIs

The USART driver provides channel-specific instances. To use this driver, access APIs by using function pointers to each instance. The list of the USART driver instances, examples of instance declaration, and the APIs contained in the instance, examples of access to the USART driver are shown in Table 2-2, Figure 2-2, Table 2-3, and Figure 2-3 to Figure 2-6.

Table 2-2 List of USART Driver Instances

Instance	Description
ARM_DRIVER_USART Driver_USART0	Instance for using SCI0
ARM_DRIVER_USART Driver_USART1	Instance for using SCI1
ARM_DRIVER_USART Driver_USART2	Instance for using SCI2
ARM_DRIVER_USART Driver_USART3	Instance for using SCI3
ARM_DRIVER_USART Driver_USART4	Instance for using SCI4
ARM_DRIVER_USART Driver_USART5	Instance for using SCI5
ARM_DRIVER_USART Driver_USART9	Instance for using SCI9

```
#include "R_Driver_USART.h"

// USART driver instance ( SCI0 )
extern ARM_DRIVER_USART Driver_USART0;
ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
```

Figure 2-2 Example of USART Driver Instance Declaration

Table 2-3 USART Driver APIs

API	Description	Reference
Initialize	Initializes the USART driver (initializes RAM and registers interrupts to NVIC). When the DMA is used for transmission/reception, it also initializes the DMA.	4.1.3
Uninitialize	Releases the USART driver (releases the pins). It will also cause a transition to the module stop state if the USART is not in the state. It will also release the DMA driver if the DMA is used for transmission/reception.	4.1.4
PowerControl	Releases the USART from the module stop state or causes a transition to the mode.	4.1.5
Send	Starts transmission.	4.1.6
Receive	Starts reception.	4.1.7
Transfer	Starts transmission/reception.	4.1.8
GetTxCount	Obtains transmission count.	4.1.9
GetRxCount	Obtains reception count.	4.1.10
Control	Executes a control command of the USART. For the control commands, see " Table 2-4".	4.1.11
GetStatus	Obtains the status of the USART.	4.1.12
SetModemControl	Controls the modem.	4.1.13
GetModemStatus	Obtains the status of the modem.	4.1.14
GetVersion	Obtains the version of the USART driver.	4.1.1
GetCapabilities	Obtains the functions of the USART driver.	4.1.2

Table 2-4 List of Control Commands

Command	Description
ARM_USART_MODE_ASYNCHRONOUS	Initializes the USART in asynchronous mode. Specify this command together with definitions of data bit length, parity function, stop bit length, and flow control. Specify a baud rate as the second argument.
ARM_USART_MODE_SYNCHRONOUS_MASTER	Initializes the USART in clock synchronous master mode. Specify this command together with definitions of flow control, clock polarity, and clock phase. Specify a baud rate as the second argument.
ARM_USART_MODE_SYNCHRONOUS_SLAVE	Initializes the USART in clock synchronous slave mode. Specify this command together with definitions of flow control, clock polarity, and clock phase.
ARM_USART_MODE_SMART_CARD	Initializes the USART in smart card mode. Specify this command together with parity function definition. Specify a baud rate as the second argument.
ARM_USART_SET_DEFAULT_TX_VALUE	Sets the transmit data (default data) to be output during reception in clock synchronous mode. Set the default data value as the second argument.
ARM_USART_SET_SMART_CARD_CLOCK	Allows or stops smart card clock output. Setting the current baud rate as the second argument enables clock output and setting 0 stops clock output.
ARM_USART_CONTROL_SMART_CARD_NACK	Enables NACK output in smart card mode. Only "1" (enable) is valid as the second argument.
ARM_USART_CONTROL_TX	Enables or disables transmission (Notes 2 and 3). Set "1" (enable) or "0" (disable) as the second argument.
ARM_USART_CONTROL_RX	Enables or disables reception (Notes 2 and 3). Set "1" (enable) or "0" (disable) as the second argument.
ARM_USART_CONTROL_TX_RX	Enables or disables transmission/reception (Notes 2 and 3). Set "1" (enable) or "0" (disable) as the second argument.
ARM_USART_ABORT_SEND	Aborts transmission.
ARM_USART_ABORT_RECEIVE	Aborts reception.
ARM_USART_ABORT_TRANSFER	Aborts transmission/reception.
ARM_USART_MODE_SINGLE_WIRE	Disabled (Note 1)
ARM_USART_MODE_IRDA	Disabled (Note 1)
ARM_USART_SET_IRDA_PULSE	Disabled (Note 1)
ARM_USART_SET_SMART_CARD_GUARD_TIME	Disabled (Note 1)
ARM_USART_CONTROL_BREAK	Disabled (Note 1)

Note 1. Not supported by the USART driver. If this definition is specified using the Control function, ARM_DRIVER_ERROR_UNSUPPORTED will be returned.

Note 2. Use ARM_USART_CONTROL_TX_RX for simultaneous transmission and reception in clock synchronous mode. This is because TE and RE need to be enabled simultaneously because of hardware restrictions. If ARM_USART_CONTROL_TX and ARM_USART_CONTROL_RX are set separately, only the one that is set first will be effective.

Note 3. If transmission and reception are set to be enabled while they are disabled, the pins to be used for the USART will also be set.
Similarly, if transmission and reception are set to be disabled while they are enabled, the pins to be used for the USART will also be released.

All the functions required for initializing the USART in asynchronous mode, clock synchronous mode, or smart card mode should be specified when setting the mode by using the Control function.

Example:

Initializes the USART in asynchronous mode with 8-bit data, no parity bit, one stop bit, no flow control, and 9600 bps.

```
sci0Drv -> Control(ARM_USART_MODE_ASYNCHRONOUS |    ARM_USART_DATA_BITS_8 |
                   ARM_USART_PARITY_NONE, | ARM_USART_STOP_BITS_1 |
                   ARM_USART_FLOW_CONTROL_NONE,    9600);
```

Table 2-5 to Table 2-10 list the functions that can be specified with the USART, and Figure 2-3 to Figure 2-6 show the coding examples for accessing the USART in each mode. If the function is not specified, the default value will be effective for the function.

Table 2-5 List of Data Length Definitions (effective in asynchronous mode)

Definition	Description
ARM_USART_DATA_BITS_7	Data length: 7 bits
ARM_USART_DATA_BITS_8 (default)	Data length: 8 bits
ARM_USART_DATA_BITS_9	Data length: 9 bits
ARM_USART_DATA_BITS_5	Disabled (Note)
ARM_USART_DATA_BITS_6	Disabled (Note)

Note. Not supported by the USART driver.

Table 2-6 List of Parity Definitions (effective in asynchronous and smart card modes)

Definition	Description
ARM_USART_PARITY_NONE,(default)	No parity (Note)
ARM_USART_PARITY_EVEN	Even parity
ARM_USART_PARITY_ODD	Odd parity

Note. ARM_USART_PARITY_NONE, cannot be used in smart card mode. If specified, ARM_USART_ERROR_PARITY will be returned.

Table 2-7 List of Stop Bit Length Definitions (effective in asynchronous mode)

Definition	Description
ARM_USART_STOP_BITS_1 (default)	One stop bit
ARM_USART_STOP_BITS_2	Two stop bits
ARM_USART_STOP_BITS_1_5	Disabled (Note)
ARM_USART_STOP_BITS_0_5	Disabled (Note)

Note. Not supported by the USART driver.

Table 2-8 List of Flow Control Definitions (effective in asynchronous and clock synchronous modes)

Definition	Description
ARM_USART_FLOW_CONTROL_NONE, (default)	No flow control
ARM_USART_FLOW_CONTROL_RTS	RTS flow control
ARM_USART_FLOW_CONTROL_CTS	CTS flow control
ARM_USART_FLOW_CONTROL_CTS_RTS	Disabled (Note)

Note. Not supported by the USART driver.

Table 2-9 List of Clock Polarity Definitions (effective in clock synchronous mode)

Definition	Description
ARM_USART_CPOL0 (default)	Clock polarity not reversed
ARM_USART_CPOL1	Clock polarity reversed

Table 2-10 List of Clock Phase Delay Definitions (effective in clock synchronous mode)

Definition	Description
ARM_USART_CPHA0 (default)	Clock phase not delayed
ARM_USART_CPHA1	Clock phase delayed

```
#include "R_Driver_USART.h"

static void usart_callback (uint32_t event);

// USART driver instance ( SCI0 )
extern ARM_DRIVER_USART Driver_USART0;
ARM_DRIVER_USART *sci0Drv = &Driver_USART0;

// Receive Buffer

static uint8_t tx_data[3] = {0x01, 0x02, 0x03};
static uint8_t rx_data[3];
main()
{

    (void)sci0Drv->Initialize(usart_callback);             /* Initialize the USART driver */
    (void)sci0Drv->PowerControl(ARM_POWER_FULL);     /* Release the USART from module stop
state */
    (void)sci0Drv->Control(ARM_USART_MODE_ASYNCHRONOUS |   /* Asynchronous mode */
                           ARM_USART_DATA_BITS_8       |   /* Data length: 8 bits */
                           ARM_USART_PARITY_NONE,      |   /* No parity */
                           ARM_USART_STOP_BITS_2       |   /* 2 stop bits */
                           ARM_USART_FLOW_CONTROL_NONE,,   /* No flow control */
                           9600);                          /* Transfer rate: 9600 bps */
    (void)sci0Drv->Control(ARM_USART_CONTROL_TX_RX,1);   /* Enable transmission/reception */

    (void)sci0Drv->Receive(&rx_data[0],3);                 /* Start 3-byte reception */
    (void)sci0Drv->Send(&tx_data[0],3);                    /* Start 3-byte transmission */

    while(1);
}

/*****************************************************************************************
* callback function
*****************************************************************************************/
static void usart_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_USART_EVENT_SEND_COMPLETE:
        {
         /* Describe the processing to be performed when transmission terminates normally */
        }
        break;

        case ARM_USART_EVENT_RECEIVE_COMPLETE:
        {
            /* Describe the processing to be performed when reception terminates normally */
        }
        break;

        default:
        {
            /* Describe the processing to be performed when a communication error occurs */
        }
        break;
    }

}   /* End of function usart_callback() */
```

Figure 2-3 Example of Access to USART Driver (Asynchronous Mode)

```
#include "R_Driver_USART.h"

static void usart_callback (uint32_t event);

// USART driver instance ( SCI0 )
extern ARM_DRIVER_USART Driver_USART0;
ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
// Receive Buffer
static uint8_t tx_data[3] = {0x01, 0x02, 0x03};
static uint8_t rx_data[3];

main()
{
    (void)sci0Drv->Initialize(usart_callback);       /* Initialize the USART driver */
    (void)sci0Drv->PowerControl(ARM_POWER_FULL);    /* Release the USART
                                                       from module stop state */
    (void)sci0Drv->Control(ARM_USART_MODE_SYNCHRONOUS_MASTER |/* Clock synchronous master */
                        ARM_USART_CPOL0 |                    /* Clock polarity not reversed */
                        ARM_USART_CPHA0 |                    /* Clock phase not delayed */
                        ARM_USART_FLOW_CONTROL_NONE,;,       /* No flow control */
                        100000);                             /* Transfer rate: 100 kbps */
    (void)sci0Drv->Control(ARM_USART_CONTROL_TX_RX,1);   /* Enable transmission/reception */

    (void)sci0Drv->Transfer (&tx_data[0], &rx_data[0], 3);
                                                  /* Start 3-byte transmission/reception */

    while(1);
}
/*****************************************************************************************
* callback function
*****************************************************************************************/
static void usart_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_USART_EVENT_SEND_COMPLETE:
        {
         /* Describe the processing to be performed when transmission terminates normally */
        }
        break;

        case ARM_USART_EVENT_RECEIVE_COMPLETE:
        {
            /* Describe the processing to be performed when reception terminates normally */
        }
        break;

        case ARM_USART_EVENT_TRANSFER_COMPLETE:
        {
            /* Describe the processing to be performed when transmission/reception terminates
               normally */
        }
        break;

        default:
        {
            /* Describe the processing to be performed when a communication error occurs */
        }
        break;
    }

}   /* End of function usart_callback() */
```

Figure 2-4 Example of Access to USART Driver (Clock Synchronous Master Mode)

```
#include "R_Driver_USART.h"

static void usart_callback (uint32_t event);

// USART driver instance ( SCI0 )
extern ARM_DRIVER_USART Driver_USART0;
ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
// Receive Buffer
static uint8_t tx_data[3] = {0x01, 0x02, 0x03};
static uint8_t rx_data[3];
main()
{
    (void)sci0Drv->Initialize(usart_callback);       /* Initialize the USART driver */
    (void)sci0Drv->PowerControl(ARM_POWER_FULL);     /* Release the USART
                                                        from module stop state */
    (void)sci0Drv->Control(ARM_USART_MODE_SYNCHRONOUS_SLAVE  |/* Clock synchronous slave */
                        ARM_USART_CPOL0 |                   /* Clock polarity not reversed */
                        ARM_USART_CPHA0 |                   /* Clock phase not delayed */
                        ARM_USART_FLOW_CONTROL_NONE,;,      /* No flow control */
                        0);
    (void)sci0Drv->Control(ARM_USART_CONTROL_TX_RX,1);   /* Enable transmission/reception */

    (void)sci0Drv->Transfer (&tx_data[0], &rx_data[0], 3);
                                                  /* Start 3-byte transmission/reception */

    while(1);
}
/*****************************************************************************************
* callback function
*****************************************************************************************/
static void usart_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_USART_EVENT_SEND_COMPLETE:
        {
         /* Describe the processing to be performed when transmission terminates normally */
        }
        break;

        case ARM_USART_EVENT_RECEIVE_COMPLETE:
        {
            /* Describe the processing to be performed when reception terminates normally */
        }
        break;

        case ARM_USART_EVENT_TRANSFER_COMPLETE:
        {
            /* Describe the processing to be performed when transmission/reception terminates
               normally */
        }
        break;

        default:
        {
            /* Describe the processing to be performed when a communication error occurs */
        }
        break;
    }

}   /* End of function usart_callback() */
```

Figure 2-5 Example of Access to USART Driver (Clock Synchronous Slave Mode)

```
#include "R_Driver_USART.h"

static void usart_callback (uint32_t event);

// USART driver instance ( SCI0 )
extern ARM_DRIVER_USART Driver_USART0;
ARM_DRIVER_USART *sci0Drv = &Driver_USART0;

// Receive Buffer
static uint8_t tx_data[3] = {0x01, 0x02, 0x03};
static uint8_t rx_data[3];

main()
{
    (void)sci0Drv->Initialize(usart_callback);       /* Initialize the USART driver */
    (void)sci0Drv->PowerControl(ARM_POWER_FULL);     /* Release the USART
                                                        from module stop state */
    (void)sci0Drv->Control(ARM_USART_MODE_SMART_CARD |     /* Smart card mode */
                       ARM_USART_PARITY_EVEN,              /* Even parity */
                       9600);                              /* Transfer rate: 9600bps */
    (void)sci0Drv->Control(ARM_USART_CONTROL_TX_RX,1);   /* Enable transmission/reception */

    (void)sci0Drv->Send(&tx_data[0],3);   /* Start 3-byte transmission/reception */

    while(1);
}

/******************************************************************************************
* callback function
******************************************************************************************/
static void usart_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_USART_EVENT_SEND_COMPLETE:
        {
         /* Describe the processing to be performed when transmission terminates normally */
        }
        break;

        case ARM_USART_EVENT_RECEIVE_COMPLETE:
        {
            /* Describe the processing to be performed when reception terminates normally */
        }
        break;

        default:
        {
            /* Describe the processing to be performed when a communication error occurs */
        }
        break;
    }

}   /* End of function usart_callback() */
```

Figure 2-6 Example of Access to USART Driver (Smart Card Mode)

2.3 Pin Configuration

The pins to be used by this driver are set and released with the R_SCI_Pinset_CHn (n = 0 to 5, 9) and R_SCI_Pinclr_CHn functions in pin.c. The R_SCI_Pinset_CHn function is called when transmission or reception is enabled by the Control function. The R_SCI_Pinclr_CHn function is called when transmission and reception are disabled by the Control function, PowerControl function or Uninitialize function.

Select the pin to be used by editing the codes in the R_SCI_Pinset_CHn and R_SCI_Pinclr_CHn functions of pin.c.

2.4 Communication Control and NVIC Interrupt Setting

The USART driver uses the interrupt handling process by default for transmission control (writing transmit data to the transmit buffer) and reception control (storing receive data to the specified receive buffer). Changing the set values of the transmission/reception control definition in r_usart_cfg.h allows the use of the DMAC or DTC to control transmission and reception.

Table 2-11 lists the definitions to set the SCIn transmission and reception control methods, and Table 2-12 lists the definitions of the transmission/reception control methods.

Table 2-11 Definitions to Set Transmission/Reception Control Methods (n = 0 to 5 and 9)

Definition	Initial Value	Description
SCIn_TRANSMIT_CONTROL	SCI_USED_INTERRUPT	SCIn transmission control (initial value: interrupt)
SCIn_RECEIVE_CONTROL	SCI_USED_INTERRUPT	SCIn reception control (initial value: interrupt)

Table 2-12 Definitions of Transmission/Reception Control Methods

Definition	Value	Description
SCI_USED_INTERRUPT	(0)	Controls transmission/reception using interrupts
SCI_USED_DMAC0	(1<<0)	Controls transmission/reception using DMAC0
SCI_USED_DMAC1	(1<<1)	Controls transmission/reception using DMAC1
SCI_USED_DMAC2	(1<<2)	Controls transmission/reception using DMAC2
SCI_USED_DMAC3	(1<<3)	Controls transmission/reception using DMAC3
SCI_USED_DTC	(1<<15)	Controls transmission/reception using DTC

It is necessary to register the interrupts used for communication control to the nested vectored interrupt controller (hereinafter referred to as NVIC) in r_system_cfg.h. For details, refer to" Setting Interrupts (NVIC)" in the RE01 1500KB, 256KB Group Getting Started Guide to Development Using CMSIS Package.

Table 2-13 shows the definition of NVIC registration for each intended use and Figure 2-7 shows the coding example for registering the interrupts to the NVIC.

Table 2-13 Definitions of NVIC Registration for Each Intended Use (n = 0 to 5, 9, and m = 0 to 3)

Mode	Intended Use	NVIC Registration Definition
Asynchronous	Transmission only	[When interrupts or DTC is used for transmission control] SYSTEM_CFG_EVENT_NUMBER_SCIn_TXI [When DMAC is used for transmission control] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
	Reception only	[When interrupts or DTC is used for reception control] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI [When DMAC is used for reception control] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI (Note) SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SCIn_ERI
	Transmission and reception	[When interrupts or DTC is used for transmission control] SYSTEM_CFG_EVENT_NUMBER_SCIn_TXI [When DMAC is used for transmission control] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		[When interrupts or DTC is used for reception control] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI [When DMAC is used for reception control] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI (Note) SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SCIn_ERI
Clock synchronous	Transmission only	[When interrupts or DTC is used for transmission control] SYSTEM_CFG_EVENT_NUMBER_SCIn_TXI [When DMAC is used for transmission control] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
	Transmission and reception, or reception only	[When interrupts or DTC is used for transmission control] SYSTEM_CFG_EVENT_NUMBER_SCIn_TXI [When DMAC is used for transmission control] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		[When interrupts or DTC is used for reception control] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI [When DMAC is used for reception control] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI (Note) SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SCIn_ERI
Smart card	Reception only	[When interrupts or DTC is used for reception control] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI [When DMAC is used for reception control] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI (Note) SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SCIn_ERI
	Transmission and reception	[When interrupts or DTC is used for transmission control] SYSTEM_CFG_EVENT_NUMBER_SCIn_TXI [When DMAC is used for transmission control] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		[When interrupts or DTC is used for reception control] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI [When DMAC is used for reception control] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI (Note) SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SCIn_ERI

Note. Even when DMAC is used, the received data is discarded in the interrupt handling process to acquire data before reception starts.

```
...
#define SYSTEM_CFG_EVENT_NUMBER_GPT_UVWEDGE
        (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)      /*!< Numbers 0/4/8/12/16/20/24/28 only */
#define SYSTEM_CFG_EVENT_NUMBER_SCI0_RXI
          (SYSTEM_IRQ_EVENT_NUMBER0)              /*!< Numbers 0/4/8/12/16/20/24/28 only */
#define SYSTEM_CFG_EVENT_NUMBER_SCI0_AM
           (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 0/4/8/12/16/20/24/28 only */
...
#define SYSTEM_CFG_EVENT_NUMBER_GPT2_CCMPB
         (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)      /*!< Numbers 1/5/9/13/17/21/25/29 only */
#define SYSTEM_CFG_EVENT_NUMBER_SCI0_TXI
          (SYSTEM_IRQ_EVENT_NUMBER1)              /*!< Numbers 1/5/9/13/17/21/25/29 only */
#define SYSTEM_CFG_EVENT_NUMBER_SPI0_SPTI
          (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)     /*!< Numbers 1/5/9/13/17/21/25/29 only */
...
#define SYSTEM_CFG_EVENT_NUMBER_GPT2_UDF
          (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)    /*!< Numbers 3/7/11/15/19/23/27/31 only */
#define SYSTEM_CFG_EVENT_NUMBER_SCI0_ERI
          (SYSTEM_IRQ_EVENT_NUMBER3)              /*!< Numbers 3/7/11/15/19/23/27/31 only */
#define SYSTEM_CFG_EVENT_NUMBER_SPI0_SPEI
          (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)     /*!< Numbers 3/7/11/15/19/23/27/31 only */
...
```

Figure 2-7 Example of Interrupt Registration to NVIC in r_system_cfg.h (SCI0 used)

2.5 Macro and Type Definitions

For the USART driver, the macro and types that can be referenced by the user are defined in the Driver_USART.h and R_Driver_USART.h files.

2.5.1 USART Control Command Definitions

The USART control commands contain the USART mode and function definitions used as the first arguments of the Control function.

Each control command is a combination of the function setting definition, and/or data length setting definition, parity setting definition, stop bit length setting definition, clock polarity (CPOL) setting definition, and clock phase delay (CPHA) setting definitions. When using function setting to set the USART communication mode, also set the data length, parity, stop bit length, clock polarity, and clock phase.

Figure 2-8 shows the structure of the USART control command containing various definitions, and Table 2-14 to Table 2-20 show the setting definitions of each function.

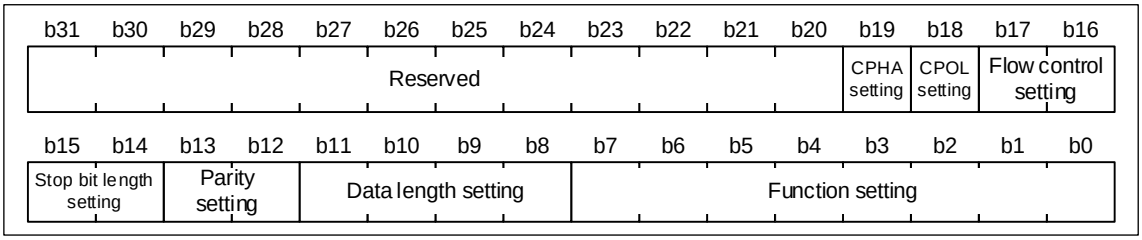


Figure 2-8 Structure of USART Control Command Containing Definitions

Table 2-14 List of USART Control Command Function Setting Definitions

Definition	Value	Description
ARM_USART_MODE_ASYNCHRONOUS	(0x01UL << ARM_USART_CONTROL_Pos)	Initializes the USART in asynchronous mode. (Note 2)
ARM_USART_MODE_SYNCHRONOUS_MASTER	(0x02UL << ARM_USART_CONTROL_Pos)	Initializes the USART in clock synchronous master mode. (Note 3)
ARM_USART_MODE_SYNCHRONOUS_SLAVE	(0x03UL << ARM_USART_CONTROL_Pos)	Initializes the USART in clock synchronous slave mode. (Note 3)
ARM_USART_MODE_SINGLE_WIRE	(0x04UL << ARM_USART_CONTROL_Pos)	Disabled (Note 1)
ARM_USART_MODE_IRDA	(0x05UL << ARM_USART_CONTROL_Pos)	Disabled (Note 1)
ARM_USART_MODE_SMART_CARD	(0x06UL << ARM_USART_CONTROL_Pos)	Initializes the USART in smart card mode. (Note 4)
ARM_USART_SET_DEFAULT_TX_VALUE	(0x10UL << ARM_USART_CONTROL_Pos)	Sets default data.
ARM_USART_SET_SMART_CARD_CLOCK	(0x13UL << ARM_USART_CONTROL_Pos)	Allows or stops smart card clock output.
ARM_USART_CONTROL_SMART_CARD_NACK	(0x14UL << ARM_USART_CONTROL_Pos)	Enables NACK output in smart card mode.
ARM_USART_CONTROL_TX	(0x15UL << ARM_USART_CONTROL_Pos)	Enables or disables transmission.
ARM_USART_CONTROL_RX	(0x16UL << ARM_USART_CONTROL_Pos)	Enables or disables reception.
ARM_USART_CONTROL_TX_RX	(0x1BUL << ARM_USART_CONTROL_Pos)	Enables or disables transmission and reception.
ARM_USART_ABORT_SEND	(0x18UL << ARM_USART_CONTROL_Pos)	Aborts transmission.
ARM_USART_ABORT_RECEIVE	(0x19UL << ARM_USART_CONTROL_Pos)	Aborts reception.
ARM_USART_ABORT_TRANSFER	(0x1AUL << ARM_USART_CONTROL_Pos)	Aborts transmission and reception.
ARM_USART_SET_IRDA_PULSE	(0x11UL << ARM_USART_CONTROL_Pos)	Disabled (Note 5)
ARM_USART_SET_SMART_CARD_GUARD_TIME	(0x12UL << ARM_USART_CONTROL_Pos)	Disabled (Note 5)
ARM_USART_CONTROL_BREAK	(0x17UL << ARM_USART_CONTROL_Pos)	Disabled (Note 5)

Note 1. This definition is not supported by the USART driver. If it is specified by the Control function, ARM_USART_ERROR_MODE will be returned.

Note 2. Should be set in combination with data length definition, parity definition, stop bit length definition, and flow control definition.

Note 3. Should be set in combination with flow control definition, clock polarity definition, and clock phase definition.

Note 4. Should be set in combination with parity definition.

Note 5. Not supported by the USART driver. If this definition is specified using the Control function, ARM_DRIVER_ERROR will be returned.

Table 2-15 List of USART Control Command Data Length Setting Definitions

Definition	Value	Description
ARM_USART_DATA_BITS_7	(7UL << ARM_USART_DATA_BITS_Pos)	7 bits
ARM_USART_DATA_BITS_8	(0UL << ARM_USART_DATA_BITS_Pos)	8 bits
ARM_USART_DATA_BITS_9	(1UL << ARM_USART_DATA_BITS_Pos)	9 bits
ARM_USART_DATA_BITS_5	(5UL << ARM_USART_DATA_BITS_Pos)	Disabled (Note)
ARM_USART_DATA_BITS_6	(6UL << ARM_USART_DATA_BITS_Pos)	Disabled (Note)

Note. This definition is not supported by the USART driver. If it is specified by the Control function, ARM_USART_ERROR_DATA_BITS will be returned.

Table 2-16 List of USART Control Command Parity Setting Definitions

Definition	Value	Description
ARM_USART_PARITY_NONE,	(0UL << ARM_USART_PARITY_Pos)	No parity
ARM_USART_PARITY_EVEN	(1UL << ARM_USART_PARITY_Pos)	Even parity
ARM_USART_PARITY_ODD	(2UL << ARM_USART_PARITY_Pos)	Odd parity

Table 2-17 List of USART Control Command Stop Bit Length Setting Definitions

Definition	Value	Description
ARM_USART_STOP_BITS_1	(0UL << ARM_USART_STOP_BITS_Pos)	1 stop bit
ARM_USART_STOP_BITS_2	(1UL << ARM_USART_STOP_BITS_Pos)	2 stop bits
ARM_USART_STOP_BITS_1_5	(2UL << ARM_USART_STOP_BITS_Pos)	Disabled (Note)
ARM_USART_STOP_BITS_0_5	(3UL << ARM_USART_STOP_BITS_Pos)	Disabled (Note)

Note. This definition is not supported by the USART driver. If it is specified by the Control function, ARM_USART_ERROR_STOP_BITS will be returned.

Table 2-18 List of USART Control Command Flow Control Setting Definitions

Definition	Value	Description
ARM_USART_FLOW_CONTROL_NONE,	(0UL << ARM_USART_FLOW_CONTROL_Pos)	No flow control
ARM_USART_FLOW_CONTROL_RTS	(1UL << ARM_USART_FLOW_CONTROL_Pos)	RTS control
ARM_USART_FLOW_CONTROL_CTS	(2UL << ARM_USART_FLOW_CONTROL_Pos)	CTS control
ARM_USART_FLOW_CONTROL_RTS_CTS	(3UL << ARM_USART_FLOW_CONTROL_Pos)	Disabled (Note)

Note. This definition is not supported by the USART driver. If it is specified by the Control function, ARM_USART_ERROR_FLOW_CONTROL will be returned.

Table 2-19 List of USART Control Command Clock Polarity Setting Definitions

Definition	Value	Description
ARM_USART_CPOL0	(0UL << ARM_USART_CPOL_Pos)	Clock polarity not reversed
ARM_USART_CPOL1	(1UL << ARM_USART_CPOL_Pos)	Clock polarity reversed

Table 2-20 List of USART Control Command Clock Phase Delay Setting Definitions

Definition	Value	Description
ARM_USART_CPHA0	(0UL << ARM_USART_CPHA_Pos)	Clock phase not delayed
ARM_USART_CPHA1	(1UL << ARM_USART_CPHA_Pos)	Clock phase delayed

2.5.2 USART-Specific Error Code Definitions

These are the error code definitions specific to the USART.

Table 2-21 List of USART-Specific Error Code Definitions

Definition	Value	Description
ARM_USART_ERROR_MODE	(ARM_DRIVER_ERROR_SPECIFIC - 1)	The specified mode is not supported.
ARM_USART_ERROR_BAUDRATE	(ARM_DRIVER_ERROR_SPECIFIC - 2)	The specified baud rate is not supported.
ARM_USART_ERROR_DATA_BITS	(ARM_DRIVER_ERROR_SPECIFIC - 3)	The specified data length is not supported.
ARM_USART_ERROR_PARITY	(ARM_DRIVER_ERROR_SPECIFIC - 4)	The specified parity is not supported.
ARM_USART_ERROR_STOP_BITS	(ARM_DRIVER_ERROR_SPECIFIC - 5)	The specified stop bit length is not supported.
ARM_USART_ERROR_FLOW_CONTROL	(ARM_DRIVER_ERROR_SPECIFIC - 6)	The specified flow control is not supported.
ARM_USART_ERROR_CPOL	(ARM_DRIVER_ERROR_SPECIFIC - 7)	Unused (Note)
ARM_USART_ERROR_CPHA	(ARM_DRIVER_ERROR_SPECIFIC - 8)	Unused (Note)

Note. This error is not returned since the USART driver supports clock polarity reversal and clock phase delay.

2.5.3 Modem Control Definition

These are the modem control definitions used by the ARM_USART_SetModemControl function.

Table 2-22 List of Modem Control Definitions

Definition	Value	Description
ARM_USART_RTS_CLEAR	(0)	Deactivates the RTS output (“H”).
ARM_USART_RTS_SET	(1)	Activates the RTS output (“L”).
ARM_USART_DTR_CLEAR	(2)	Disabled (Note)
ARM_USART_DTR_SET	(3)	Disabled (Note)

Note. This definition is not supported by this driver.

2.5.4 USART Event Code Definitions

These are the definitions of events to be notified by callback functions. If multiple events occur at the same time, an ORed value will be notified. For event code identification in the callback functions, refer to the examples of API access in each mode (Figure 2-3 to Figure 2-6 in section 2.2 Driver APIs).

Table 2-23 List of USART Event Codes

Definition	Value	Description
ARM_USART_EVENT_SEND_COMPLETE	(1UL << 0)	Transmission was completed.
ARM_USART_EVENT_RECEIVE_COMPLETE	(1UL << 1)	Reception was completed.
ARM_USART_EVENT_TRANSFER_COMPLETE	(1UL << 2)	Transmission/reception was completed.
ARM_USART_EVENT_TX_COMPLETE	(1UL << 3)	Unused
ARM_USART_EVENT_TX_UNDERFLOW	(1UL << 4)	Unused
ARM_USART_EVENT_RX_OVERFLOW	(1UL << 5)	A reception overflow was generated.
ARM_USART_EVENT_RX_TIMEOUT	(1UL << 6)	Unused
ARM_USART_EVENT_RX_BREAK	(1UL << 7)	Unused
ARM_USART_EVENT_RX_FRAMING_ERROR	(1UL << 8)	A framing error was generated.
ARM_USART_EVENT_RX_PARITY_ERROR	(1UL << 9)	A parity error was generated.
ARM_USART_EVENT_CTS	(1UL << 10)	Unused
ARM_USART_EVENT_DSR	(1UL << 11)	Unused
ARM_USART_EVENT_DCD	(1UL << 12)	Unused
ARM_USART_EVENT_RI	(1UL << 13)	Unused

2.6 Structure Definitions

For the USART driver, the structures that can be referenced by the user are defined in the Driver_USART.h file.

2.6.1 ARM_USART_STATUS Structure

This structure is used when the GetStatus function returns the status of the USART.

Table 2-24 ARM_USART_STATUS Structure

Element Name	Type	Description
tx_busy	uint32_t:1	Shows transmission status. 0: Waiting for transmission 1: Transmission in progress (busy)
rx_busy	uint32_t:1	Shows reception status. 0: Waiting for reception 1: Reception in progress (busy)
tx_underflow	uint32_t:1	Unused (fixed at 0)
rx_overflow	uint32_t:1	Shows reception overflow status. 0: Reception overflow has not been generated. 1: Reception overflow has been generated.
rx_break	uint32_t:1	Unused (fixed at 0)
rx_framing_error	uint32_t:1	Shows framing error status. 0: Framing error has not been generated. 1: Framing error has been generated.
rx_parity_error	uint32_t:1	Shows parity error status. 0: Parity error has not been generated. 1: Parity error has been generated.
reserved	uint32_t:25	Reserved area

2.6.2 ARM_USART_MODEM_STATUS Structure

This structure is used when the GetModemStatus function returns the modem status.

Table 2-25 ARM_USART_MODEM_STATUS Structure

Element Name	Type	Description
cts	uint32_t:1	Shows CTS status. 0: CTS in inactive status 1: CTS in active status
dsr	uint32_t:1	Unused (fixed at 0)
dcd	uint32_t:1	Unused (fixed at 0)
ri	uint32_t:1	Unused (fixed at 0)
reserved	uint32_t:28	Reserved area

2.6.3 ARM_USART_CAPABILITIES Structure

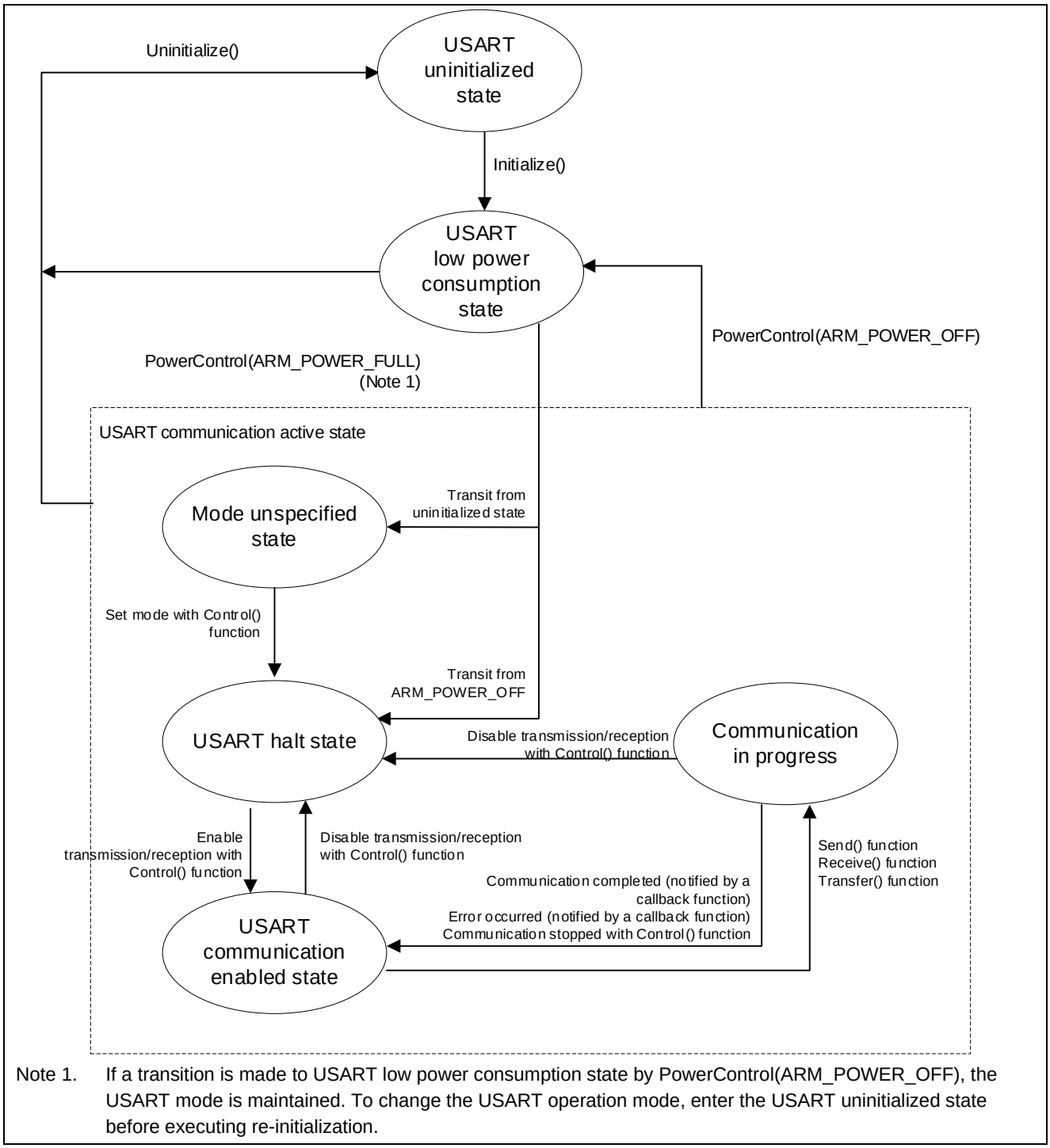
This structure is used when the GetCapabilities function returns the functions of USART.

Table 2-26 ARM_USART_CAPABILITIES Structure

Element Name	Type	Description	Value
asynchronous	uint32_t:1	Enables/disables asynchronous mode.	1 (enable)
synchronous_master	uint32_t:1	Enables/disables clock synchronous master mode.	1 (enable)
synchronous_slave	uint32_t:1	Enables/disables clock synchronous slave mode.	1 (enable)
single_wire	uint32_t:1	Enables/disables single wire mode.	0 (disable)
irda	uint32_t:1	Enables/disables IRDA mode.	0 (disable)
smart_card	uint32_t:1	Enables/disables smart card mode.	1 (enable)
smart_card_clock	uint32_t:1	Enables/disables smart card clock output.	1 (enable)
flow_control_rts	uint32_t:1	Enables/disables RTS flow control.	1 (enable)
flow_control_cts	uint32_t:1	Enables/disables CTS flow control.	1 (enable)
event_tx_complete	uint32_t:1	Enables/disables transmission end event.	0 (disable)
event_rx_timeout	uint32_t:1	Enables/disables reception timeout event.	0 (disable)
rts	uint32_t:1	Enables/disables RTS line.	1 (enable)
cts	uint32_t:1	Enables/disables CTS line.	1 (enable)
dtr	uint32_t:1	Enables/disables DTR line.	0 (disable)
dsr	uint32_t:1	Enables/disables DSR line.	0 (disable)
dcd	uint32_t:1	Enables/disables DCD line.	0 (disable)
ri	uint32_t:1	Enables/disables RI line.	0 (disable)
event_cts	uint32_t:1	Enables/disables CTS event.	0 (disable)
event_dsr	uint32_t:1	Enables/disables DSR event.	0 (disable)
event_dcd	uint32_t:1	Enables/disables DCD event.	0 (disable)
event_ri	uint32_t:1	Enables/disables RI event.	0 (disable)
reserved	uint32_t:11	Reserved area	-

2.7 State Transitions

The state transition diagram of the USART driver is shown in Figure 2-9, and state-specific events and actions are shown in Table 2-27 and Table 2-28.



Note 1. If a transition is made to USART low power consumption state by PowerControl(ARM_POWER_OFF), the USART mode is maintained. To change the USART operation mode, enter the USART uninitialized state before executing re-initialization.

Figure 2-9 State Transitions of USART Driver

Table 2-27 Events and Actions Specific to USART Driver State (Note 1)

State	Overview	Event	Action
USART uninitialized state	The USART driver is in this state after release from a reset.	Execution of Initialize() function	Enters the USART low power consumption state
USART low power consumption state	No clock is supplied to the SCI module in this state.	Execution of Uninitialize() function	Enters the USART uninitialized state
		Execution of PowerControl(ARM_POWER_FULL) function	Enters the mode unspecified state or USART halt state. (Note 2)
Mode unspecified state	USART mode is not specified in this state.	Execution of Control(ARM_USART_MODE_XXX) function (Note 3)	Enters the USART halt state.
		Execution of Uninitialize() function	Enters the USART uninitialized state.
		Execution of PowerControl(ARM_POWER_OFF) function	Enters the USART low power consumption state.
USART halt state	Communication is halted in this state.	Execution of Uninitialize() function	Enters the USART uninitialized state.
		Execution of PowerControl(ARM_POWER_OFF) function	Enters the USART low power consumption state.
		Enabling of transmission/reception with Control() function.	Enters the USART communication enabled state.
		Execution of SetModemControl() function	Controls the RTS pin. (Note 4)
USART communication enabled state	USART is waiting for communication in this state.	Execution of Uninitialize() function	Enters the USART uninitialized state.
		Execution of PowerControl(ARM_POWER_OFF) function	Enters the USART low power consumption state.
		Disabling of transmission/reception with Control() function.	Enters the USART halt state
		Execution of Send function.	Enters the communicating state (starts transmission).
		Execution of Receive function.	Enters the communicating state (starts reception).
		Execution of Transfer function.	Enters the communicating state (starts transmission/reception).
		Execution of SetModemControl() function.	Controls the RTS pin. (Note 4)

Table 2-28 Events and Actions Specific to USART Driver State (Note 1)

State	Overview	Event	Action
Communicating state	Communication is in progress in this state.	Execution of Uninitialize() function.	Enters the USART uninitialized state.
		Execution of PowerControl(ARM_POWER_OFF) function.	Enters the USART low power consumption state.
		Completion of communication.	Enters the USART communication enabled state and calls the callback function. (Note 5)
		Error occurrence	Enters the USART communication enabled state and calls the callback function. (Note 5)
		Disabling of transmission/reception with Control() function.	Enters the USART halt state
		Execution of Control(ARM_USART_ABORT_XXX) function.	Aborts communication and enters the USART communication enabled state.
		Execution of SetModemControl() function.	Controls the RTS pin. (Note 4)

Note 1. The GetVersion, GetCapabilities, GetTxCount, GetRxCount, GetStatus, and GetModemStatus functions can be executed in any state.

Note 2. Enters the mode unspecified state from the USART uninitialized state if the USART mode has not been set.

Note 3. XXX indicates one of the following modes.
ASYNCHRONOUS: Asynchronous mode
SYNCHRONOUS_MASTER: Clock synchronous master mode
SYNCHRONOUS_SLAVE : Clock synchronous slave mode
SMART_CARD: Smart card mode

Note 4. Effective only if the RTS pin has been set in r_usart_cfg.h and the hardware RTS function has been set to be disabled when the mode has been set.

Note 5. Only if a callback function is specified when the Initialize function is executed, the callback function will be called.

3. Descriptions of Driver Operations

The USART driver provides the asynchronous, clock synchronous, and smart card communications functions. This chapter describes the procedures for initializing the USART driver in each mode.

3.1 Asynchronous Mode

3.1.1 Initial Setting Procedure of Asynchronous Mode

Figure 3-1 shows the initial setting procedure of asynchronous mode.

To enable transmission and reception, register the interrupts to use to NVIC in r_system_cfg.h. For details, see section 2.3, Communication Control.

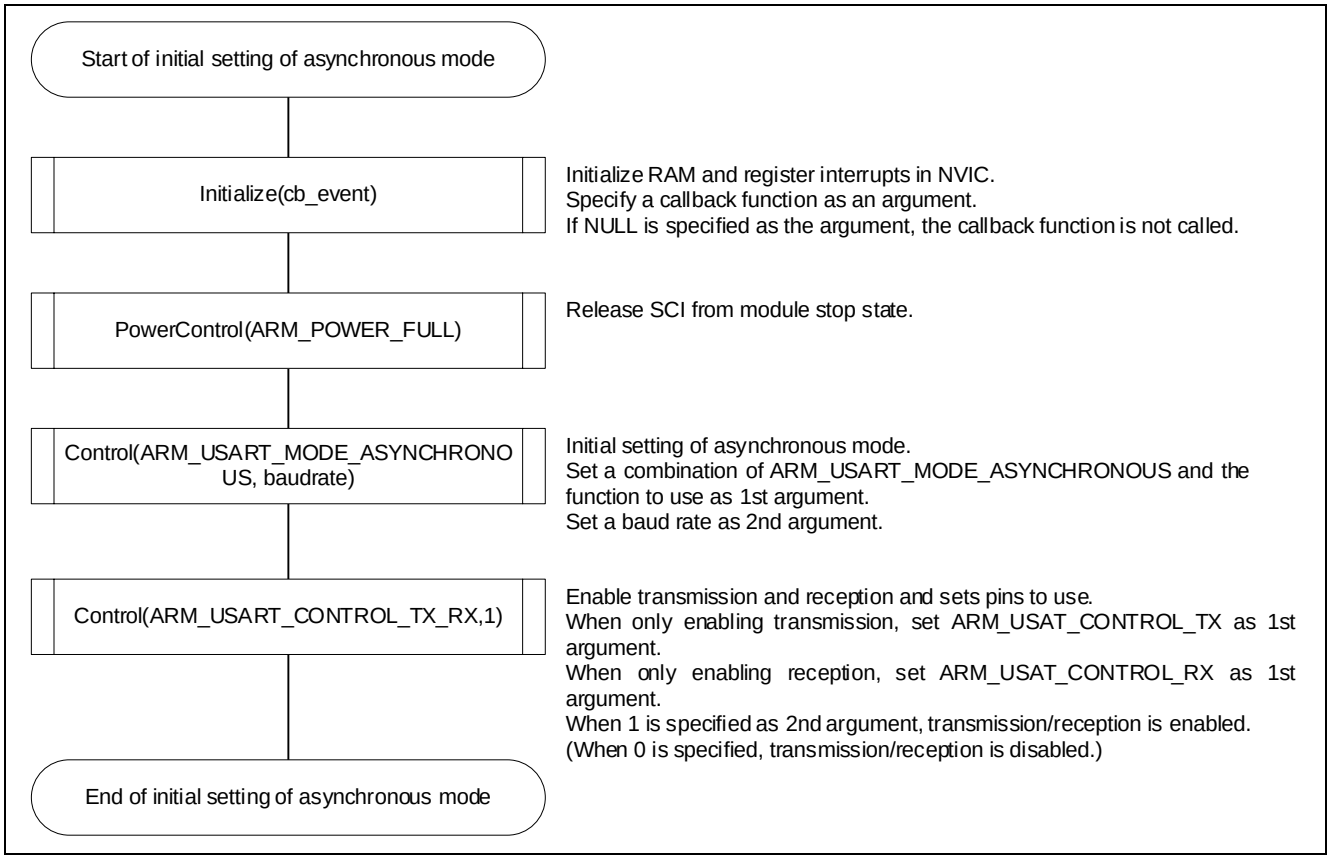


Figure 3-1 Asynchronous Mode Initialization Procedure

3.1.2 Transmission in Asynchronous Mode

Figure 3-2 shows the transmission procedure in asynchronous mode.

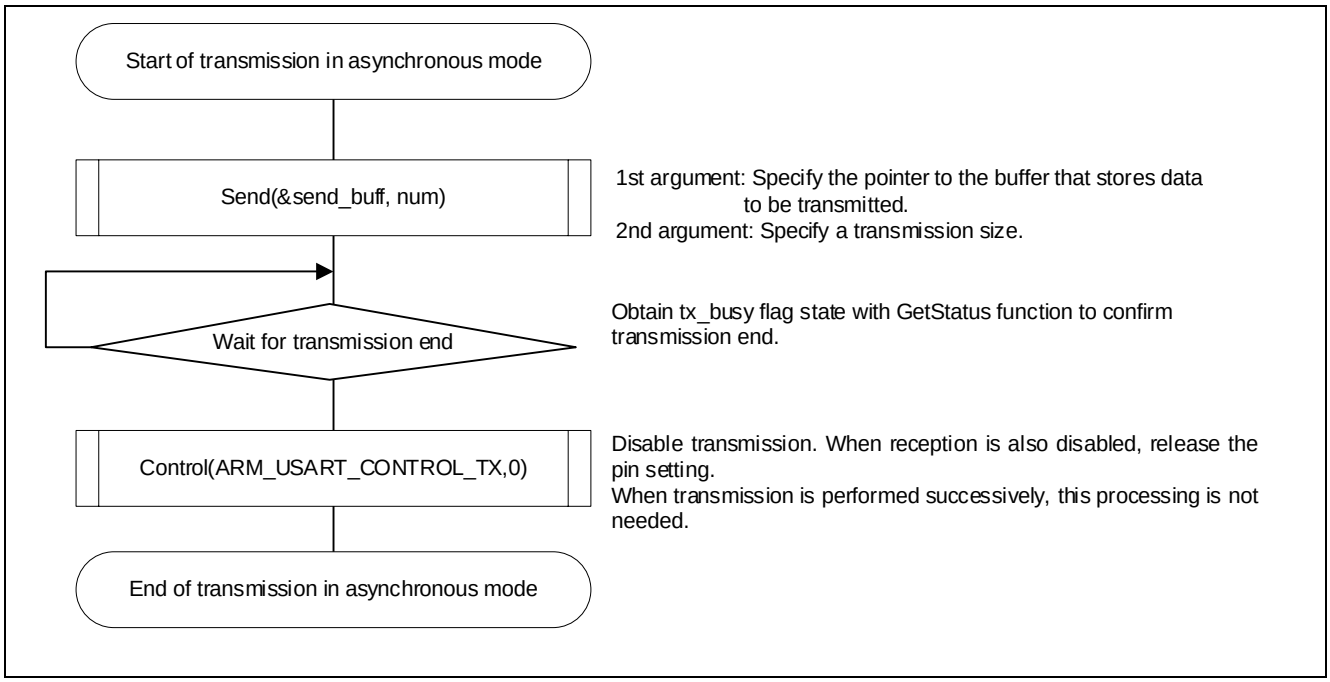


Figure 3-2 Transmission Procedure in Asynchronous Mode

If a callback function has been set, the function is called upon completion of transmission using ARM_USART_EVENT_SEND_COMPLETE as an argument.

The specific transmission operation in asynchronous mode is different depending on what method is used for communication control: the interrupt, DMAC, or DTC. FigureFigure 3-3 shows the transmission operation when the interrupt is used for communication control, Figure 3-4 shows the transmission operation when the DMAC is used for communication control, and Figure 3-5 shows the transmission operation when the DTC is used for communication control.

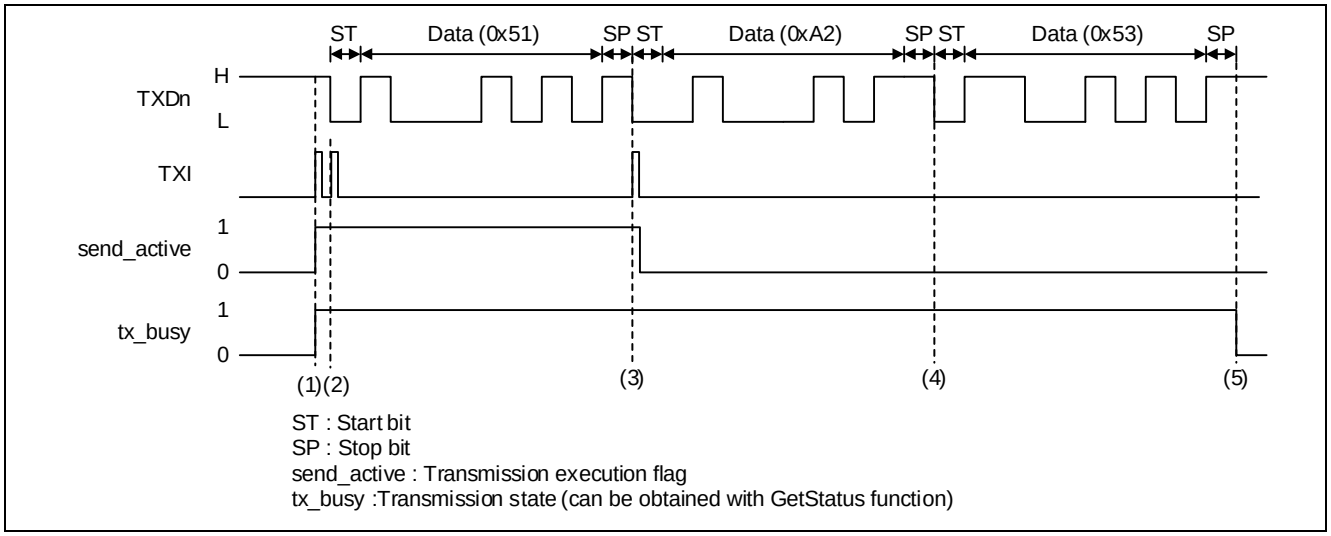


Figure 3-3 Transmission Operation using Interrupt for Control (3 bytes transmitted)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy). The TXI interrupt is also generated and the first data byte is written to the transmit data register (TDR).

(2) At the second TXI interrupt, the second transmit data byte is written to the TDR register.

(3) At the TXI interrupt after the last data is written, the TXI interrupt is disabled. If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) After the second data byte is output, the last data, which has been written in step (3) is output. (Note)

(5) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

Note. If the Send function is executed while the send_active flag is "0" and the transmit buffer holds a value (period between (3) and (4)), the Send function is terminated after the send_active flag is set to "1". When the transmit buffer becomes empty (at (4)), the TXI interrupt is generated and the first data byte is written to the TDR register.

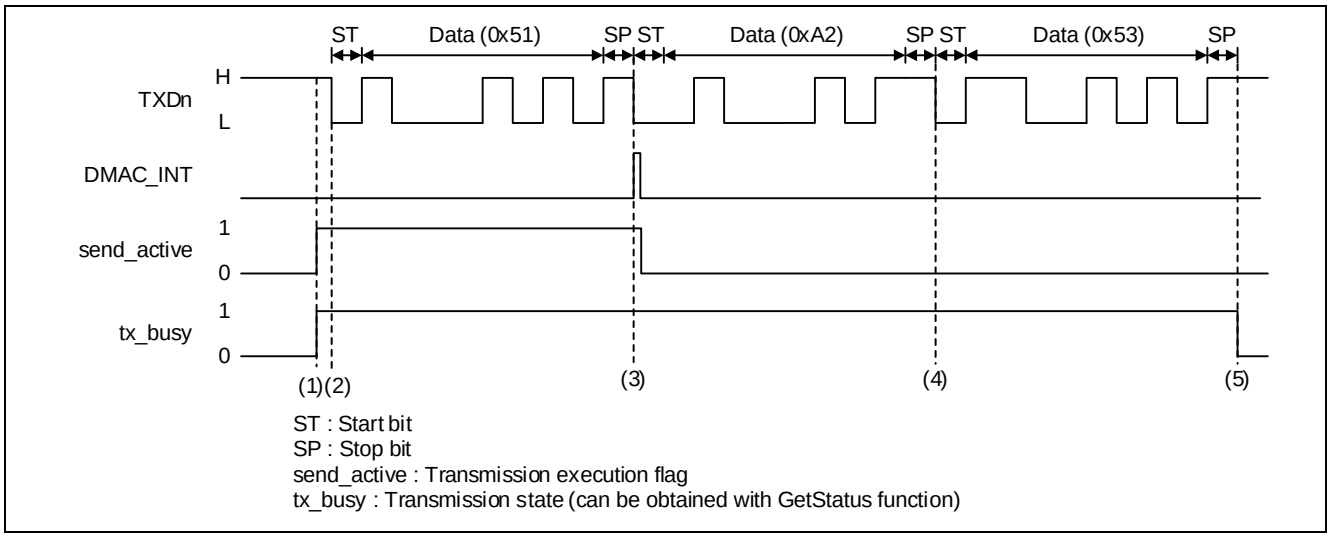


Figure 3-4 Transmission Operation using DMAC for Control (3 bytes transmitted)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy). The TXI interrupt is set as the DMAC transfer trigger and the first data byte is written to the transmit data register (TDR).

(2) The second and the following data bytes are transferred to the TDR register through DMA transfer.

(3) When the last data is transferred through DMA transfer, the DMAC transfer end interrupt is generated and the send_active flag is cleared to "0" (transmission ready). If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) After the second data byte is output, the last data, which has been written in step (3) is output. (Note)

(5) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

Note. If the Send function is executed while the tx_busy flag is "0" and the transmit buffer holds a value (period between (3) and (4)), execution is suspended until the transmit buffer becomes empty (at (4)), and step (1) is executed.
The suspension time can be changed with the value defined in SCI_CHECK_TDRE_TIMEOUT of r_usart_cfg.h. If the transmit buffer does not become empty within the specified time, the Send function returns ARM_DRIVER_ERROR_BUSY.

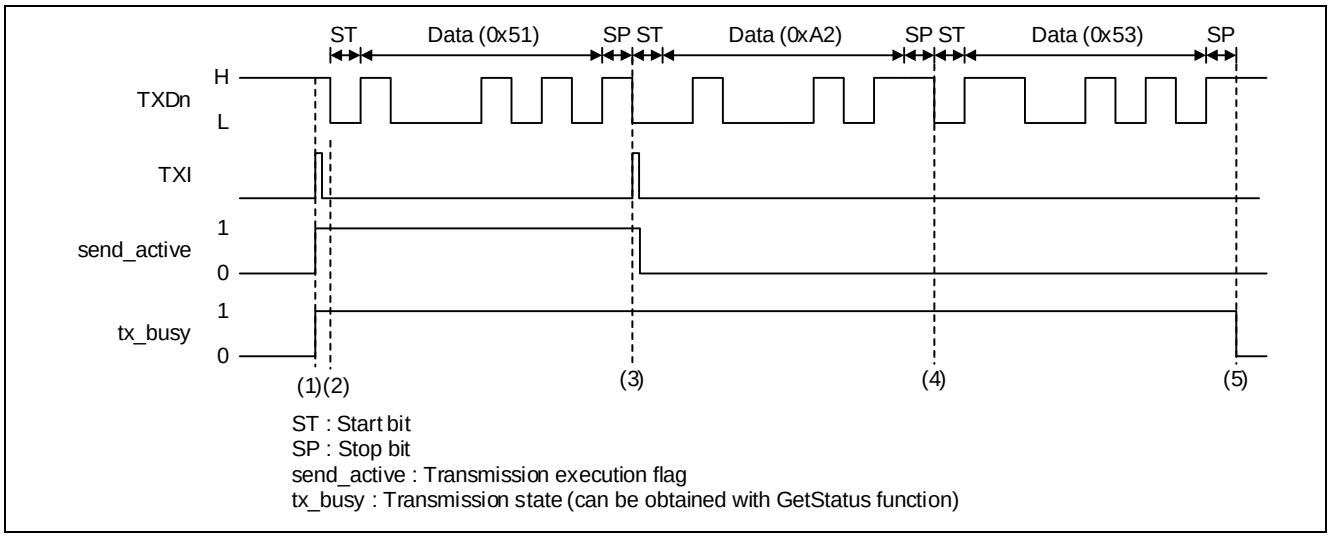


Figure 3-5 Transmission Operation using DTC for Control (3 bytes transmitted)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy), the TXI interrupt is set as the DTC transfer trigger, and the TXI interrupt is enabled. Here, the TXI interrupt is generated and the first data byte is written to the transmit data register (TDR).

(2) The second and the following data bytes are transferred to the TDR register through DMA transfer.

(3) When the last data is transferred through DMA transfer, the TXI interrupt is generated and the send_active flag is cleared to "0" (transmission ready). If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) After the second data byte is output, the last data, which has been written in step (3) is output. (Note)

(5) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end)

Note. If the Send function is executed while the tx_busy flag is "0" and the transmit buffer holds a value (period between (3) and (4)), execution is suspended until the transmit buffer becomes empty (at (4)), and step (1) is executed.
The suspension time can be changed with the value defined in SCI_CHECK_TDRE_TIMEOUT of r_usart_cfg.h. If the transmit buffer does not become empty within the specified time, the Send function returns ARM_DRIVER_ERROR_BUSY.

3.1.3 Reception in Asynchronous Mode

Figure 3-6 shows the reception procedure in asynchronous mode.

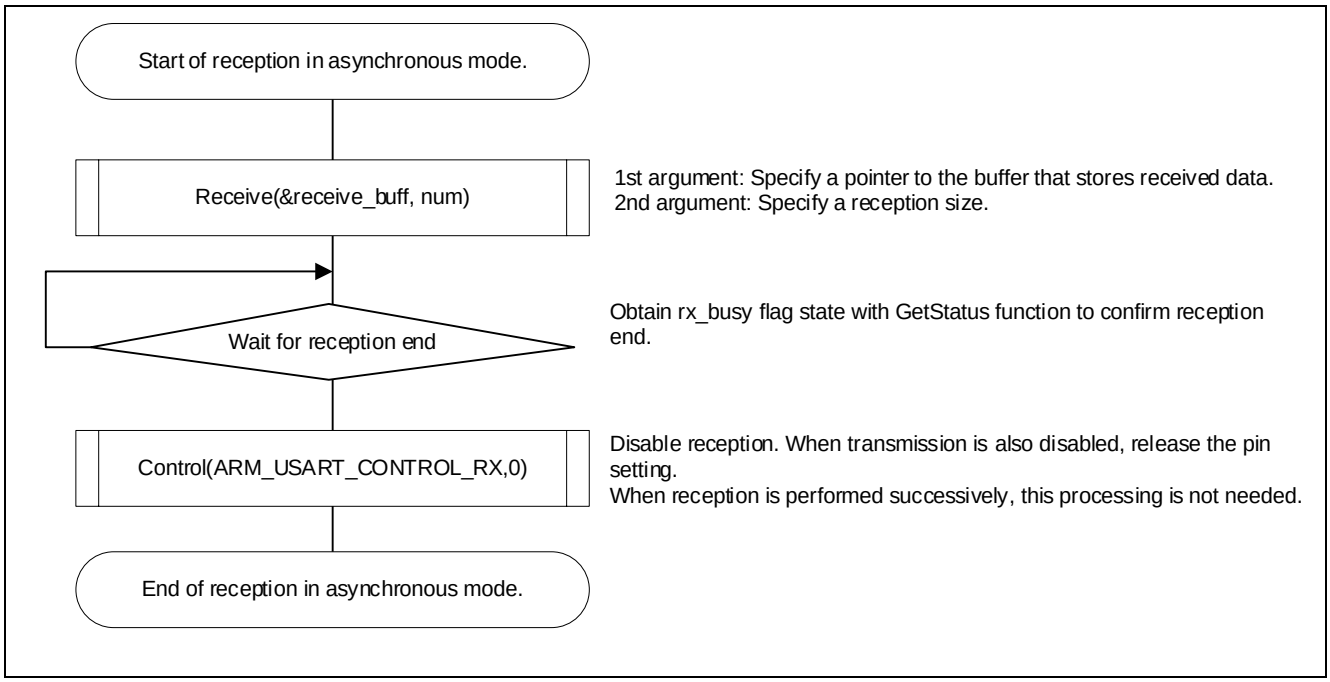


Figure 3-6 Reception Procedure in Asynchronous Mode

If a callback function has been set, the function is called upon completion of reception using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument.

If a reception error is generated, the callback function is called using the error event information as an argument, and reception is completed. Table 3-1 shows the error event information generated during reception in asynchronous mode.

Table 3-1 Error Event Information Generated during Reception in Asynchronous Mode

Error Event Information (Note)	Description
ARM_USART_EVENT_RX_OVERFLOW	An overrun error has been generated.
ARM_USART_EVENT_RX_FRAMING_ERROR	A framing error has been generated.
ARM_USART_EVENT_RX_PARITY_ERROR	A parity error has been generated.

Note. If more than one error is generated, a callback function is called using the ORed error event information as an argument.

The specific reception operation in asynchronous mode is different depending on what method is used for communication control: the interrupt, DMAC, or DTC. FigureFigure 3-7 shows the reception operation when the interrupt is used for communication control, Figure 3-8 shows the reception operation when the DMAC is used for communication control, and Figure 3-9 shows the reception operation when the DTC is used for communication control.

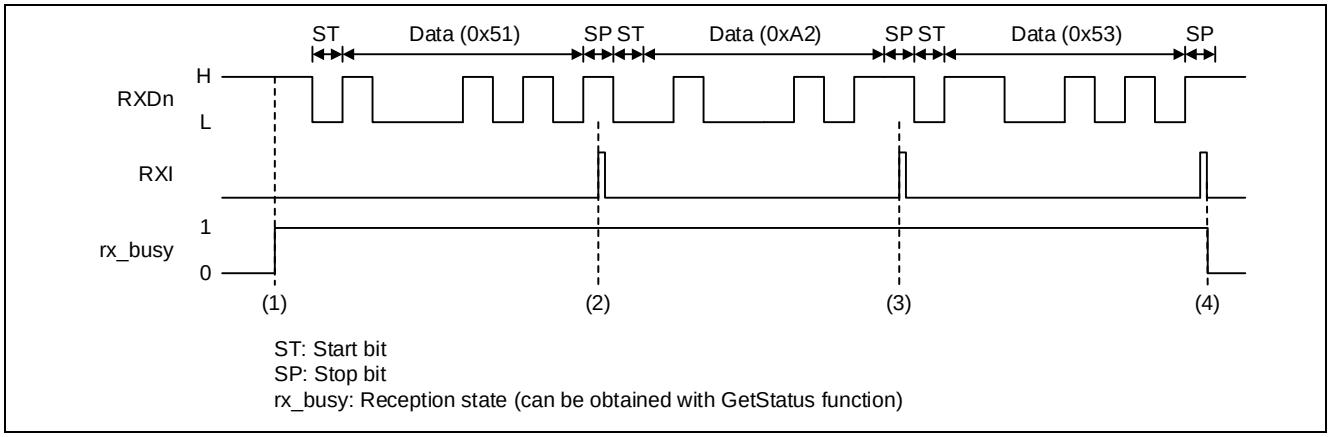


Figure 3-7 Reception Operation using Interrupt for Control (3 bytes received)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy).

(2) When data is received via the RXDn pin, the RXI interrupt is generated. At the RXI interrupt, the received data is read from the receive data register (RDR) to the specified buffer.

(3) The RXI interrupt is generated each time one byte is received, and the received data is read from the RDR register.

(4) At the RXI interrupt after the last data is read, the rx_busy flag is cleared to "0" (reception wait state). If a callback function has been registered, the function is executed using ARM_USART_EVENT_ RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

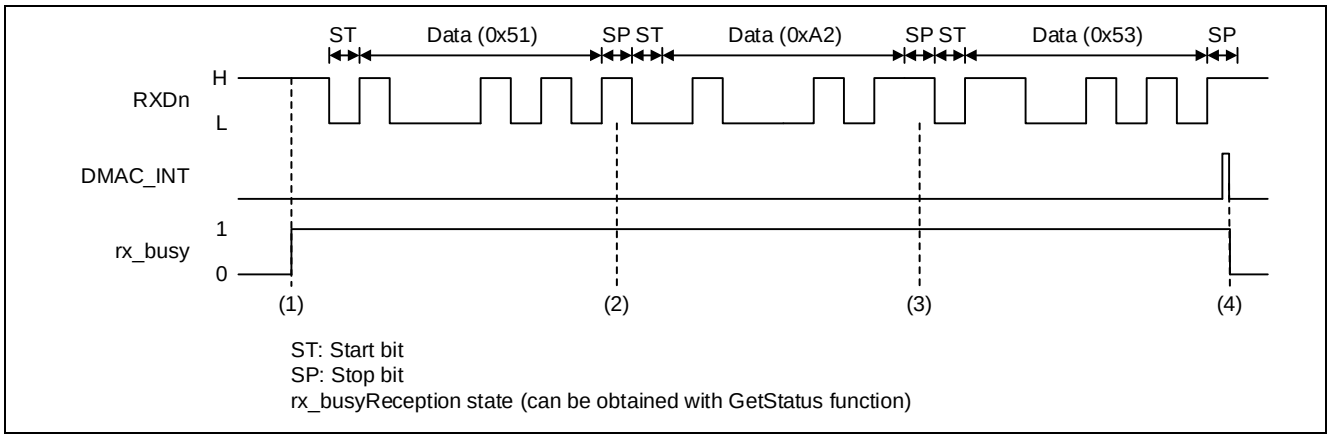


Figure 3-8 Reception Operation using DMAC for Control (3 bytes received)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy).

(2) When data is received via the RXDn pin, the received data is transferred through DMA transfer from the receive data register (RDR) to the specified buffer.

(3) DMA transfer is generated each time one byte is received, and the received data is transferred from the RDR register.

(4) After the last data is transferred, the DMAC transfer end interrupt is generated. The rx_busy flag is cleared to "0" (reception wait state) in the interrupt handling process. If a callback function has been registered, the function is executed using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

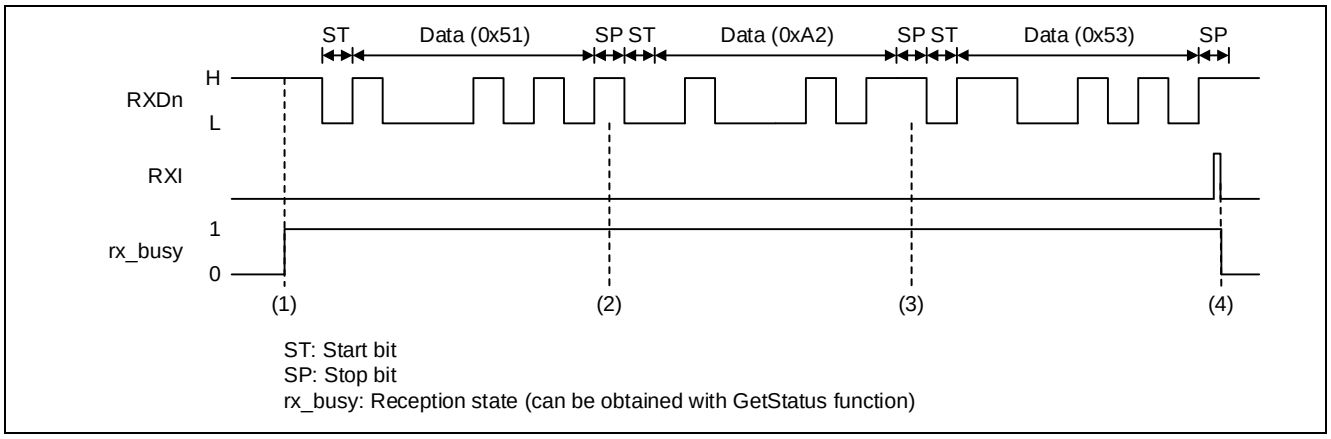


Figure 3-9 Reception Operation using DTC for Control (3 bytes received)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy).

(2) When data is received via the RXDn pin, the received data is transferred from the receive data register (RDR) to the specified buffer through DMA transfer.

(3) DMA transfer is generated each time one byte is received, and the received data is transferred from the RDR register.

(4) After the last data is transferred, the RXI interrupt is generated and the rx_busy flag is cleared to "0" (reception wait state) through the interrupt handling process. If a callback function has been registered, the function is executed using ARM_USART_EVENT_ RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

3.2 Clock Synchronous Master Mode

3.2.1 Initial Setting Procedure of Clock Synchronous Master Mode

Figure 3-10 shows the initial setting procedure of clock synchronous master mode.

To enable transmission and reception, register the interrupts to use to NVIC in r_system_cfg.h. For details, see section 2.3, Communication Control.

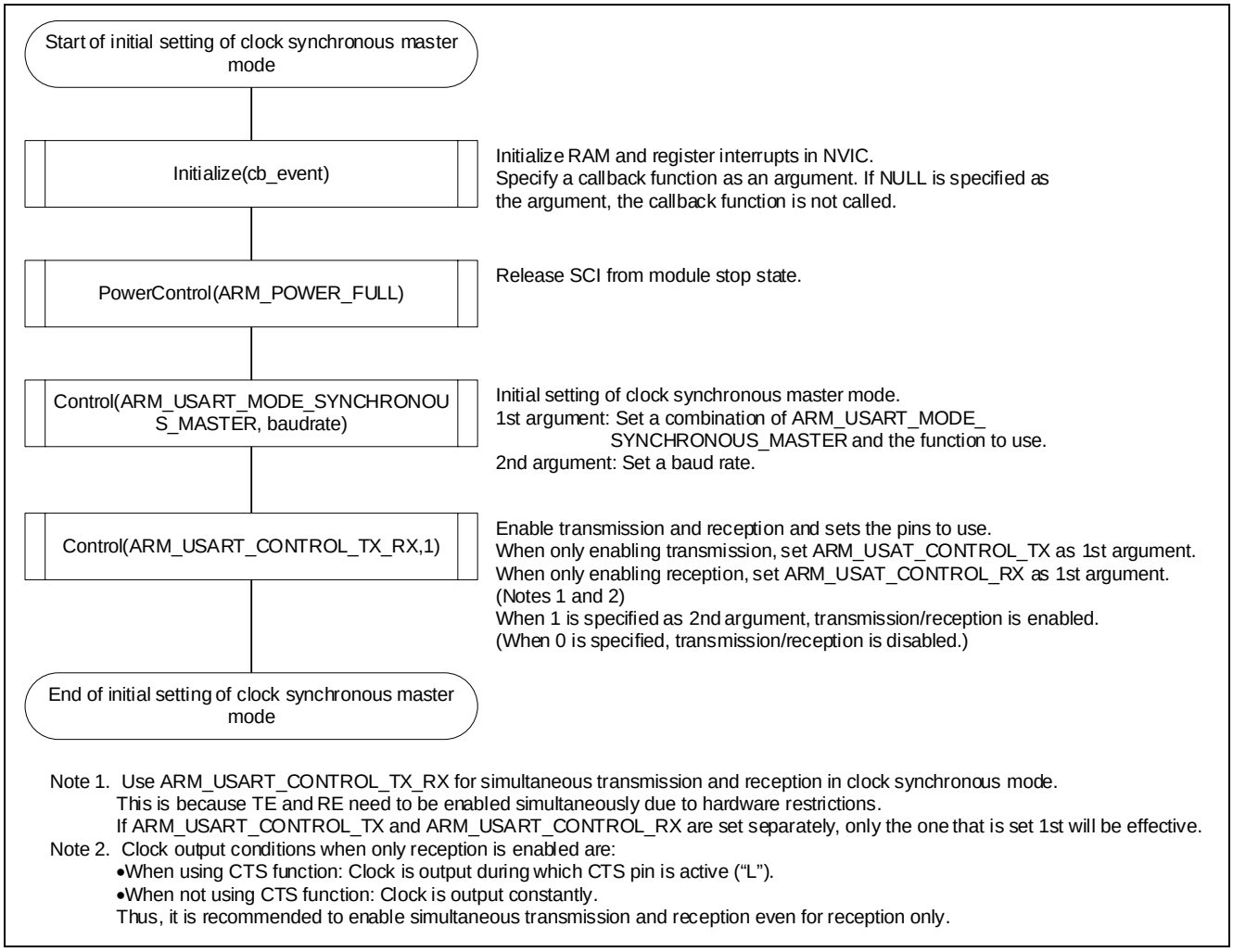


Figure 3-10 Clock Synchronous Master Mode Initialization Procedure

3.2.2 Transmission in Clock Synchronous Master Mode

Figure 3-11 shows the transmission procedure in clock synchronous master mode.

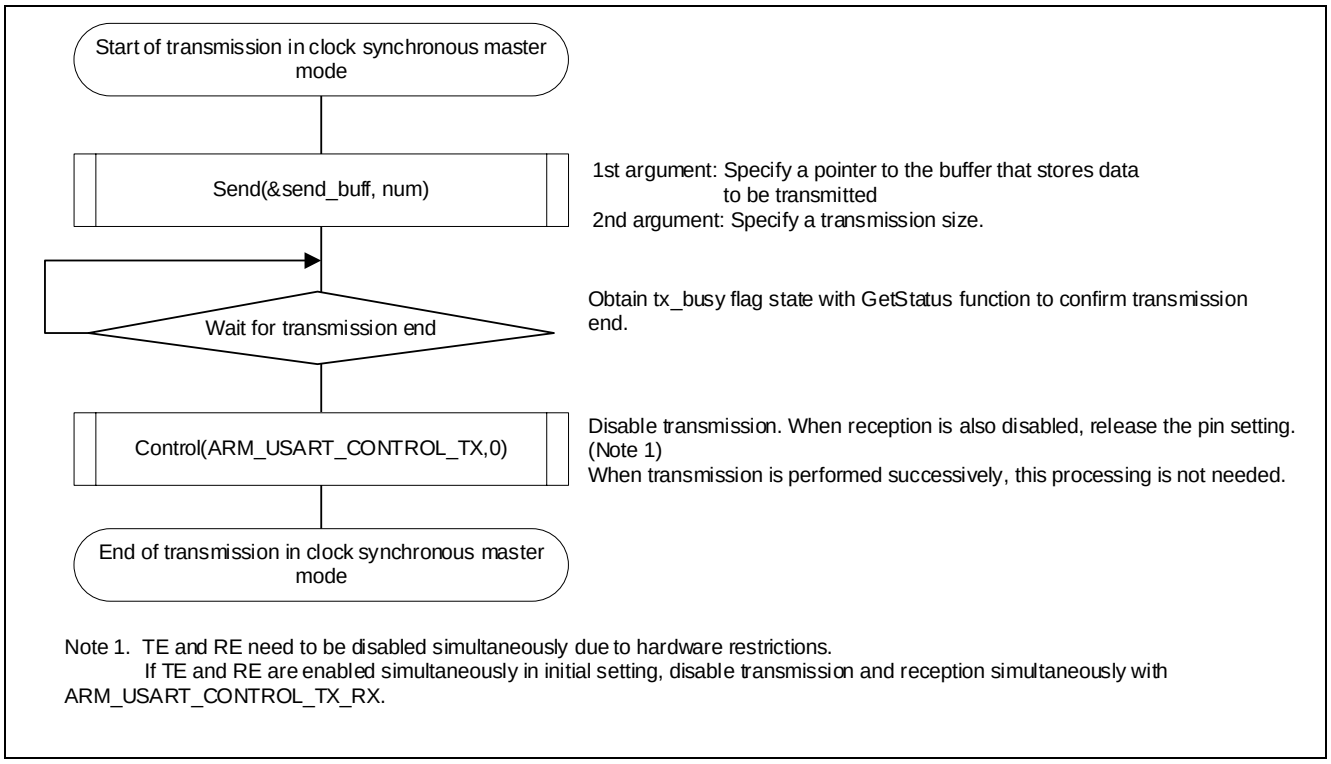


Figure 3-11 Transmission Procedure in Clock Synchronous Master Mode

If a callback function has been set, the function is called upon completion of transmission using ARM_USART_EVENT_SEND_COMPLETE as an argument.

The specific transmission operation in clock synchronous master mode is different depending on what method is used for communication control: the interrupt, DMAC, or DTC. FigureFigure 3-12 shows the transmission operation when the interrupt is used for communication control, Figure 3-13 shows the transmission operation when the DMAC is used for communication control, and Figure 3-14 shows the transmission operation when the DTC is used for communication control.

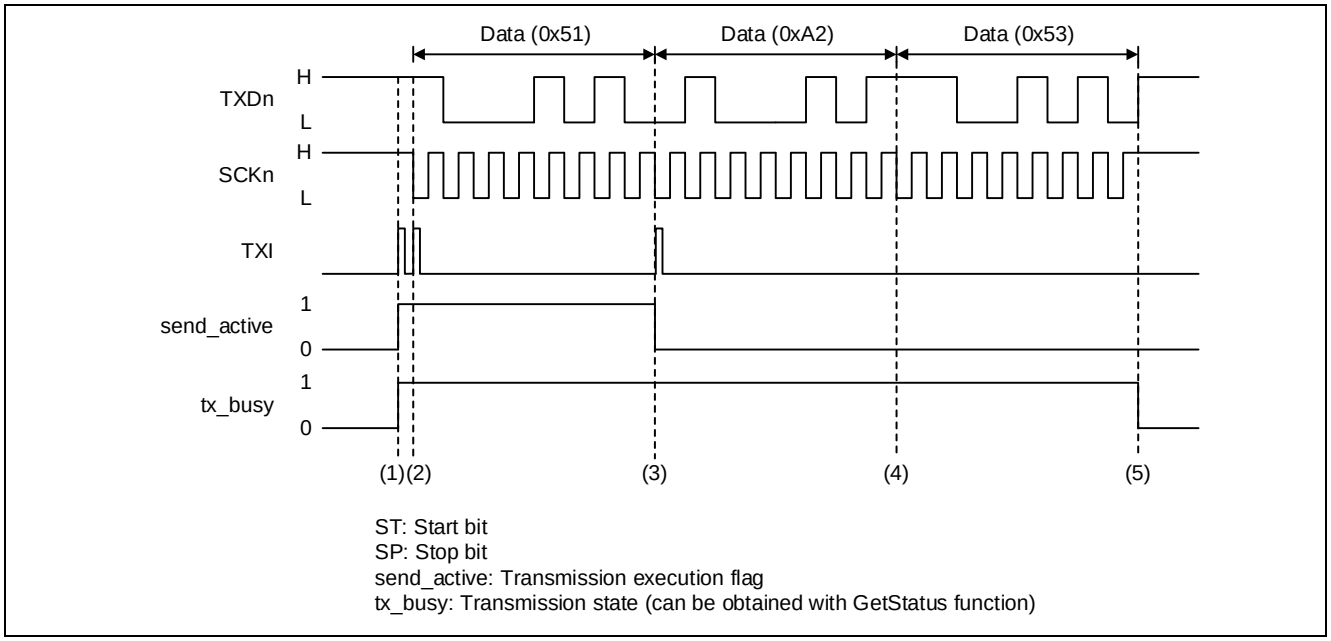


Figure 3-12 Transmission Operation using Interrupt for Control (3 bytes transmitted)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy). The TXI interrupt is also generated and the first data byte is written to the transmit data register (TDR).

(2) At the second TXI interrupt, the second transmit data byte is written to the TDR register.

(3) At the TXI interrupt after the last data is written, the TXI interrupt is disabled and the send_active flag is cleared to "0" (transmission ready). If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) After the second data byte is output, the last data, which has been written in step (3) is output. (Note)

(5) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

Note. If the Send function is executed while the tx_busy flag is "0" and the transmit buffer holds a value (period between (3) and (4)), the Send function is terminated after the tx_busy flag is set to "1". When the transmit buffer becomes empty (at (4)), the TXI interrupt is generated and the first data byte is written to the TDR register.

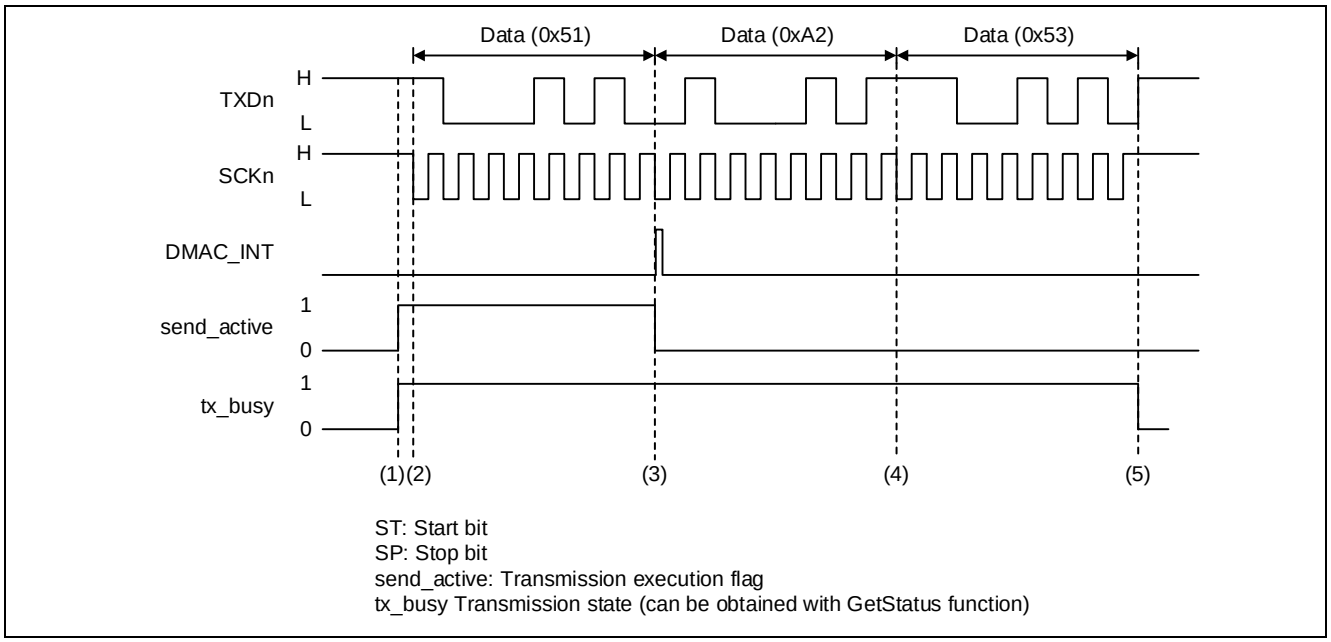


Figure 3-13 Transmission Operation using DMAC for Control (3 bytes transmitted)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy), the TXI interrupt is set as the DMAC transfer trigger, and the first data byte is written to the transmit data register (TDR).

(2) The second and the following data bytes are transferred to the TDR register through DMA transfer.

(3) When the last data is transferred through DMA transfer, the DMAC transfer end interrupt is generated and the send_active flag is cleared to "0" (transmission ready). If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) After the second data byte is output, the last data, which has been written in step (3) is output. (Note)

(5) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

Note. If the Send function is executed while the tx_busy flag is "0" and the transmit buffer holds a value (period between (3) and (4)), execution is suspended until the transmit buffer becomes empty (at (4)), and step (1) is executed.
The suspension time can be changed with the value defined in SCI_CHECK_TDRE_TIMEOUT of r_usart_cfg.h. If the transmit buffer does not become empty within the specified time, the Send function returns ARM_DRIVER_ERROR_BUSY.

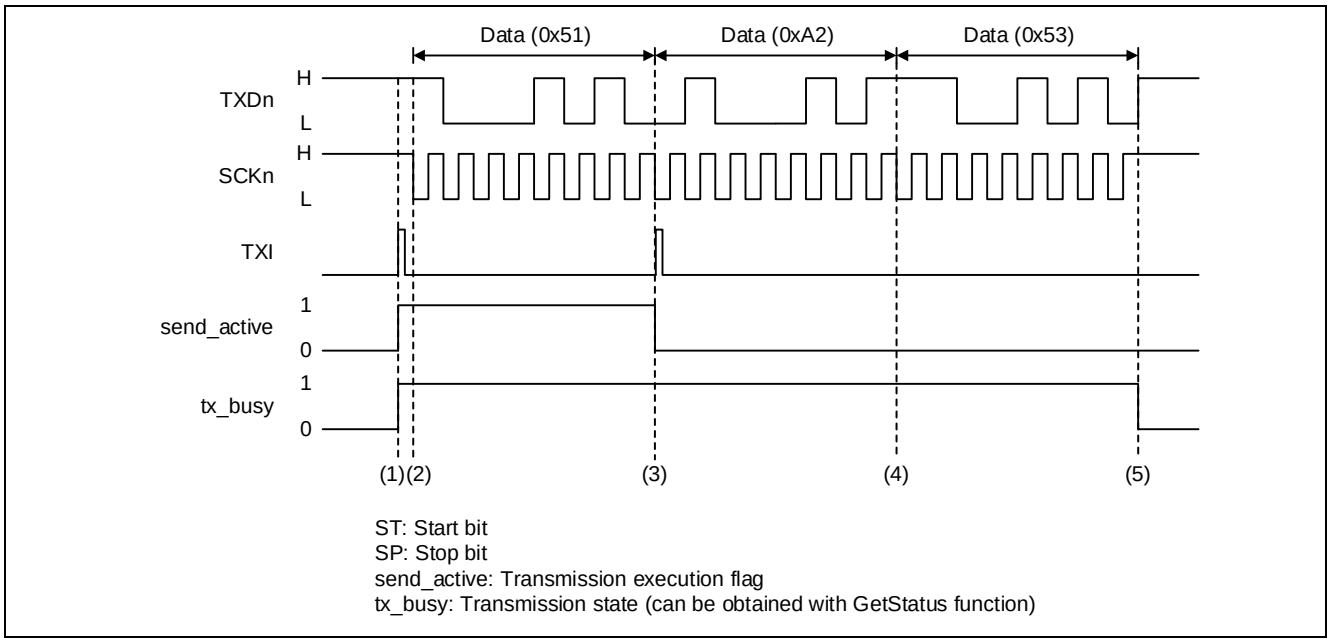


Figure 3-14 Transmission Operation using DTC for Control (3 bytes transmitted)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy), the TXI interrupt is set as the DTC transfer trigger, and the TXI interrupt is enabled. Here, the TXI interrupt is generated and the first data byte is written to the transmit data register (TDR).

(2) The second and the following data bytes are transferred to the TDR register through DMA transfer.

(3) When the last data is transferred through DMA transfer, the TXI interrupt is generated and the send_active flag is cleared to "0" (transmission ready). If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) After the second data byte is output, the last data, which has been written in step (3) is output. (Note)

(5) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

Note. the Send function is executed while the tx_busy flag is "0" and the transmit buffer holds a value (period between (3) and (4)), execution is suspended until the transmit buffer becomes empty (at (4)), and step (1) is executed.
The suspension time can be changed with the value defined in SCI_CHECK_TDRE_TIMEOUT of r_usart_cfg.h. If the transmit buffer does not become empty within the specified time, the Send function returns ARM_DRIVER_ERROR_BUSY.

3.2.3 Reception in Clock Synchronous Master Mode

Figure 3-15 shows the reception procedure in clock synchronous master mode.

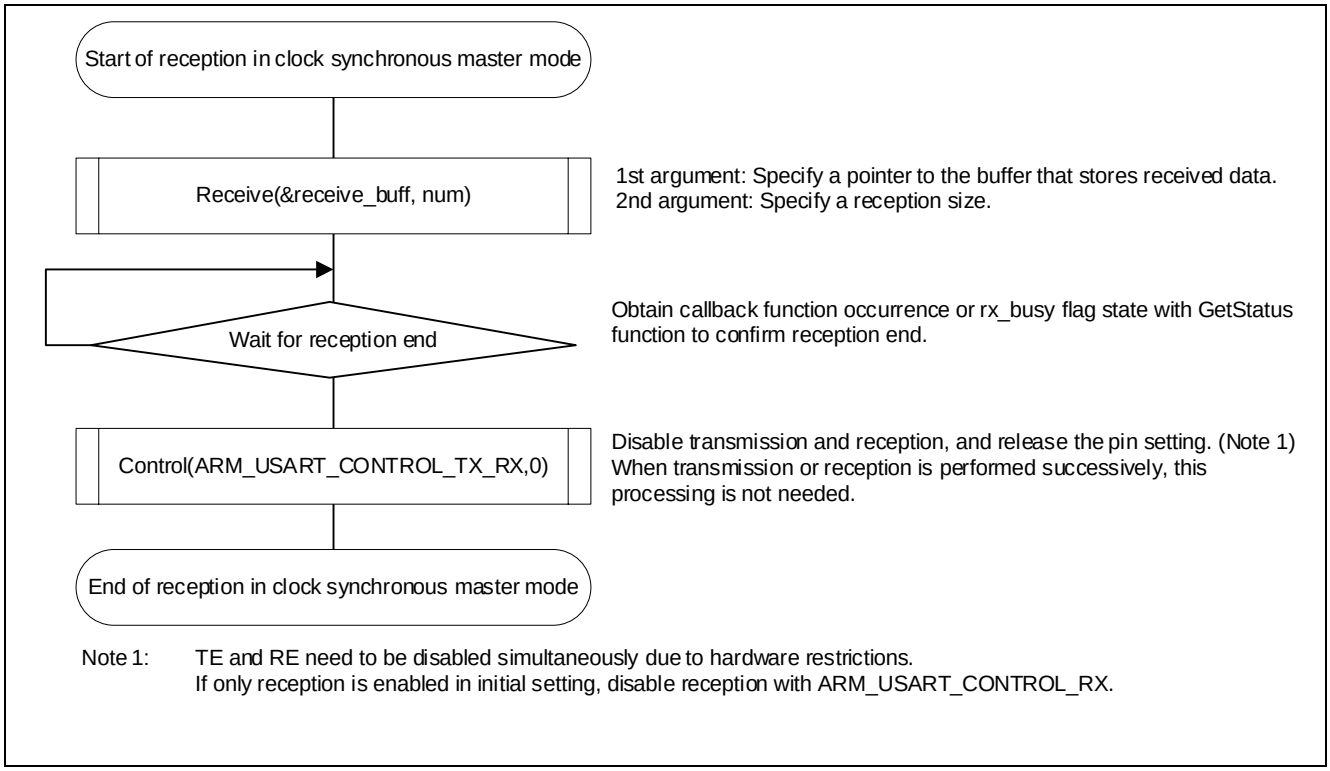


Figure 3-15 Reception Procedure in Clock Synchronous Master Mode

If a callback function has been set, the function is called upon completion of reception using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument.

If a reception error is generated, the callback function is called using the error event information as an argument, and transmission and reception are terminated. Table 3-2shows the error event information generated during reception in clock synchronous master mode.

Table 3-2 Error Event Information Generated during Reception in Clock Synchronous Master Mode

Error Event Information	Description
ARM_USART_EVENT_RX_OVERFLOW	An overrun error has been generated.

The specific reception operation in clock synchronous master mode is different depending on what method is used for communication control: the interrupt, DMAC, or DTC. In addition, dummy data is written to the transmit buffer to output the clock signal if transmission is enabled. Dummy data to be output can be changed using the ARM_USART_SET_DEFAULT_TX_VALUE command.

FigureFigure 3-16 shows the reception operation when the interrupt is used for communication control, Figure 3-17 shows the reception operation when the DMAC is used for communication control, and Figure 3-18 shows the reception operation when the DTC is used for communication control.

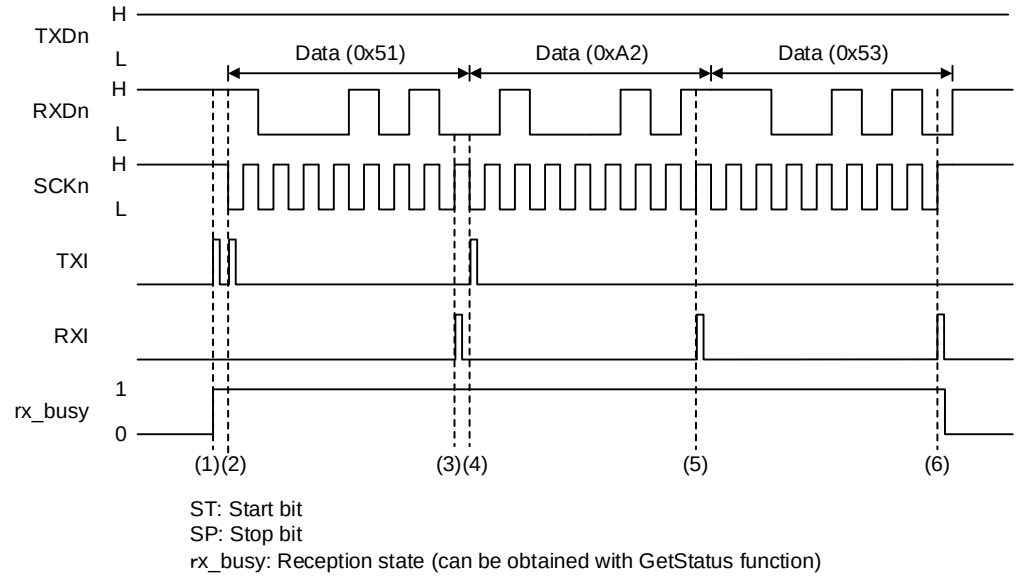


Figure 3-16 Reception Operation using Interrupt for Control
(3 bytes transmitted, transmission enabled, dummy data 0xFF)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy). The TXI interrupt is also generated and the dummy data is written to the transmit data register (TDR).

(2) At the second TXI interrupt, the second dummy data byte is written to the TDR register.

(3) When one byte has been received, the RXI interrupt is generated and the value is read from the receive data register (RDR) to the specified buffer.

(4) At the TXI interrupt after the specified number of bytes are written, the TXI interrupt is disabled.

(5) The RXI interrupt is generated each time one byte is received, and the received data is read from the RDR register.

(6) At the RXI interrupt after the last data is read, the rx_busy flag is cleared to "0" (reception wait state). If a callback function has been registered, the function is executed using ARM_USART_EVENT_ RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

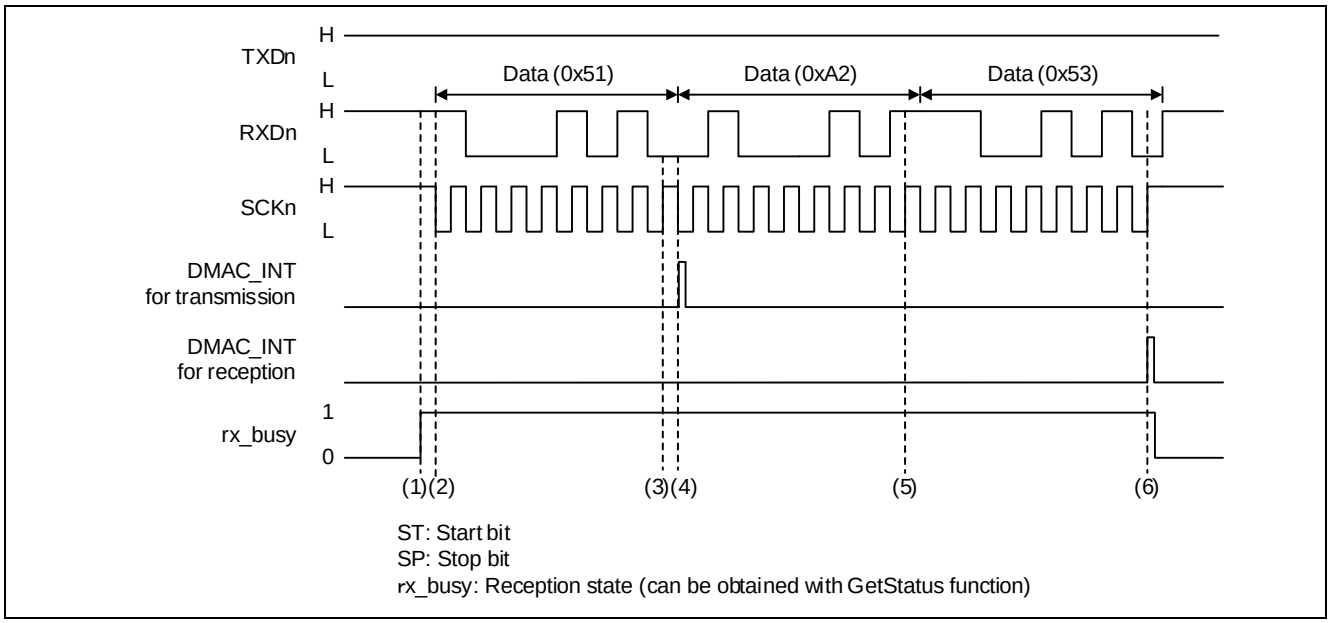


Figure 3-17 Reception Operation using DMAC for Control (3 bytes transmitted, transmission enabled, dummy data 0xFF)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy), the TXI and RXI interrupts are set as the DMAC transfer triggers, and then the first dummy data is written to the transmit data register (TDR).

(2) The second and the following dummy data bytes are transferred to the transmit data register (TDR) through DMA transfer.

(3) When data is received via the RXDn pin, the value in the receive data register (RDR) is transferred to the specified buffer through DMA transfer.

(4) When the specified number of bytes have been written, the DMAC transfer end interrupt on the transmitting side is generated.

(5) Each time one byte has been received, the RDR register value is transferred to the specified buffer through DMA transfer.

(6) After the specified size of data is transferred, the DMAC transfer end interrupt is generated on the receiving side. The rx_busy flag is cleared to "0" (reception wait state) in the interrupt handling process. If a callback function has been registered, the function is executed using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

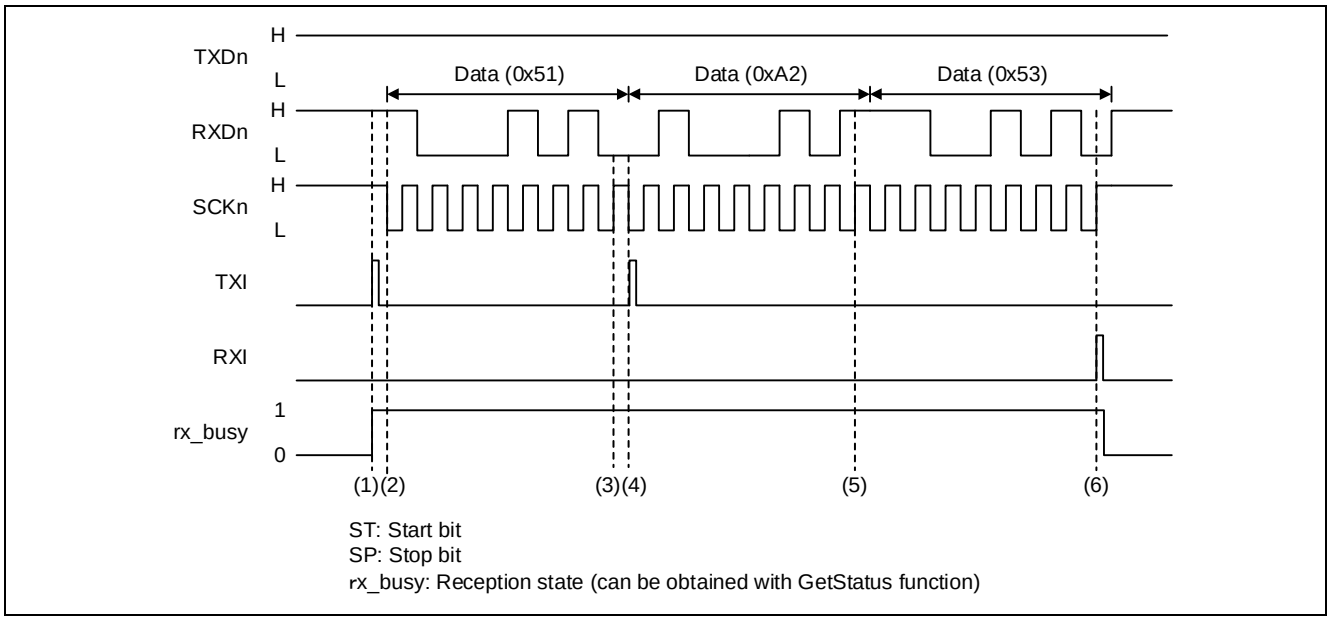


Figure 3-18 Reception Operation using DTC for Control (3 bytes transmitted, transmission enabled, dummy data 0xFF)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy), the TXI interrupt is set as the DTC transfer trigger, and the TXI interrupt is enabled. Here, the TXI interrupt is generated and the first dummy data is written to the transmit data register (TDR).

(2) The second and the following dummy data bytes are transferred to the TDR register through DMA transfer.

(3) When data is received via the RXDn pin, the value in the receive data register (RDR) is transferred to the specified buffer through DMA transfer.

(4) When the specified number of bytes have been written, the TXI interrupt is generated.

(5) Each time one byte has been received, the RDR register value is transferred to the specified buffer through DMA transfer.

(6) After the last data is transferred, the RXI interrupt is generated. The rx_busy flag is cleared to "0" (reception wait state) in the interrupt handling process. If a callback function has been registered, the function is executed using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

3.2.4 Transmission and Reception in Clock Synchronous Master Mode

Figure 3-19 shows the transmission and reception procedure in clock synchronous master mode.

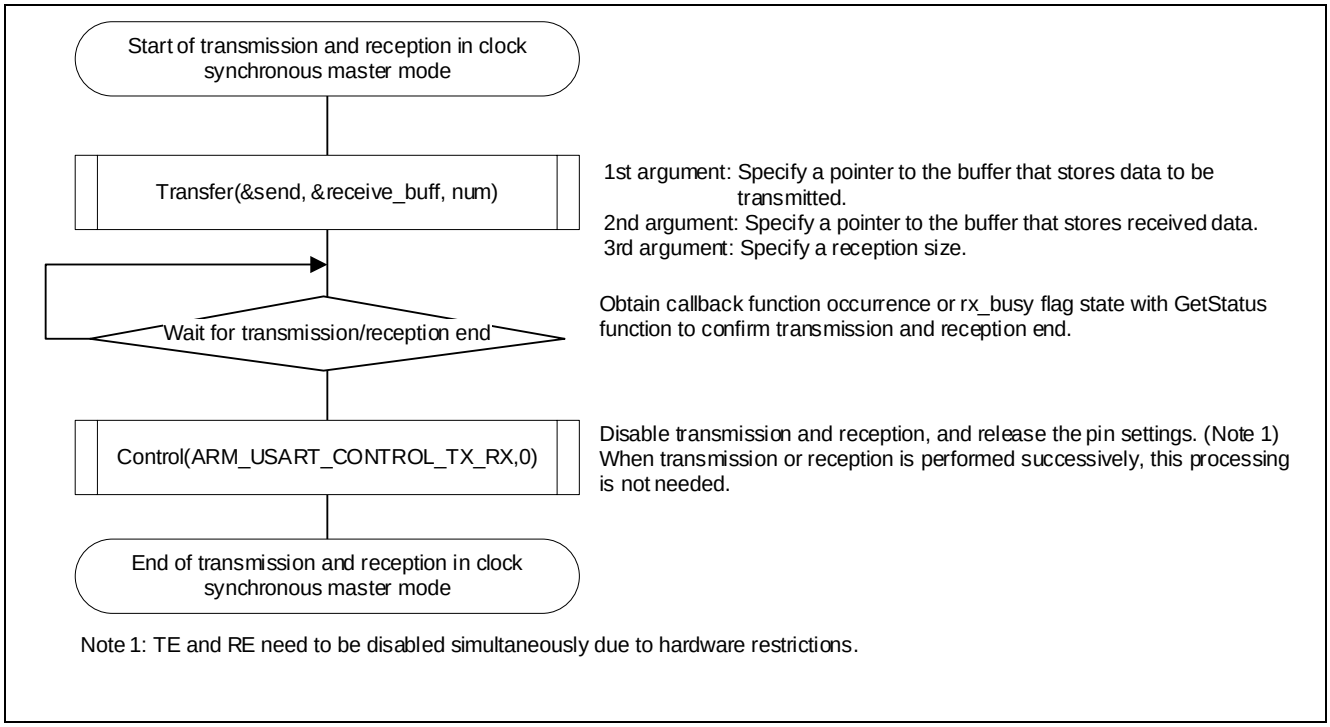


Figure 3-19 Transmission and Reception Procedure in Clock Synchronous Master Mode

If a callback function has been set, the function is called upon completion of transmission and reception using ARM_USART_EVENT_TRANSFER_COMPLETE as an argument.

If a reception error is generated, the callback function is called using the error event information as an argument, and transmission and reception are terminated. Table 3-3 shows the error event information generated during transmission and reception in clock synchronous master mode.

Table 3-3 Error Event Information Generated during Transmission and Reception in Clock Synchronous Master Mode

Error Event Information	Description
ARM_USART_EVENT_RX_OVERFLOW	An overrun error has been generated.

The specific transmission and reception operation in clock synchronous master mode is different depending on what method is used for communication control: the interrupt, DMAC, or DTC. Figure 3-20 shows the operation when the interrupt is used for communication control, Figure 3-21 shows the operation when the DMAC is used for communication control, and Figure 3-22 shows the operation when the DTC is used for communication control.

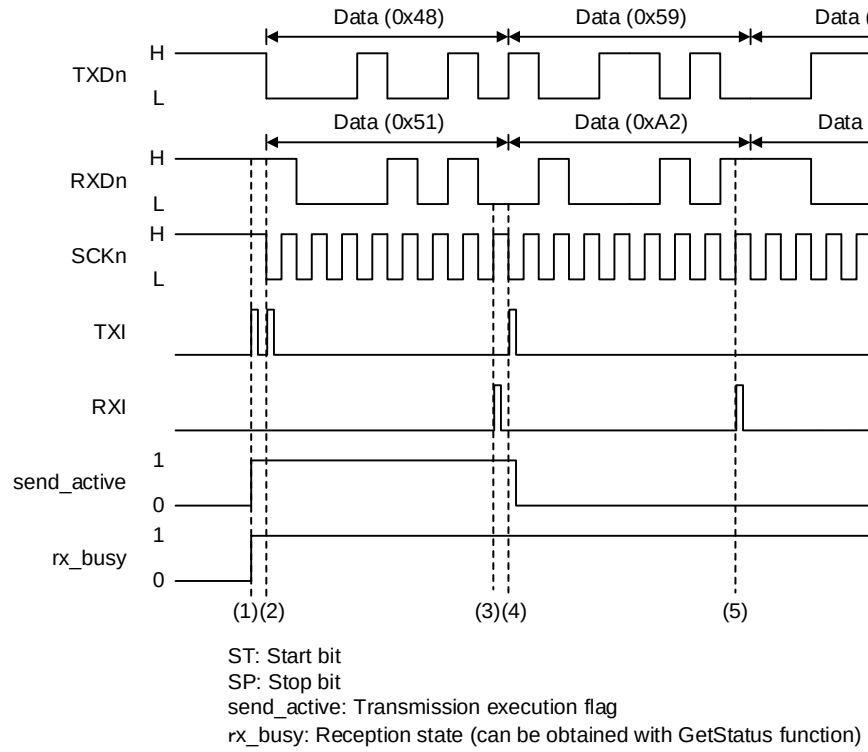


Figure 3-20 Transmission and Reception Operation using Interrupt for Control (3 bytes received)

(1) When the Transfer function is executed, the tx_busy flag is set to "1" (transmission busy), the rx_busy flag is set to "1" (reception busy), and the TXI interrupt is enabled. Here, the TXI interrupt is generated and the first transmit data byte is written to the transmit data register (TDR).

(2) At the second TXI interrupt, the second transmit data byte is written to the TDR register.

(3) When one byte has been received, the RXI interrupt is generated and the value is read from the receive data register (RDR) to the specified buffer.

(4) At the TXI interrupt after the specified number of bytes are written, the send_active flag is cleared to "0" (transmission ready), and the TXI interrupt is disabled.

(5) The RXI interrupt is generated each time one byte is received, and the received data is read from the RDR register.

(6) At the RXI interrupt after the last data is read, the rx_busy flag is cleared to "0" (reception wait state). If a callback function has been registered, the function is executed using ARM_USART_EVENT_TRANSFER_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the tx_busy and rx_busy flags are cleared to "0" (transmission wait state and reception wait state, respectively) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

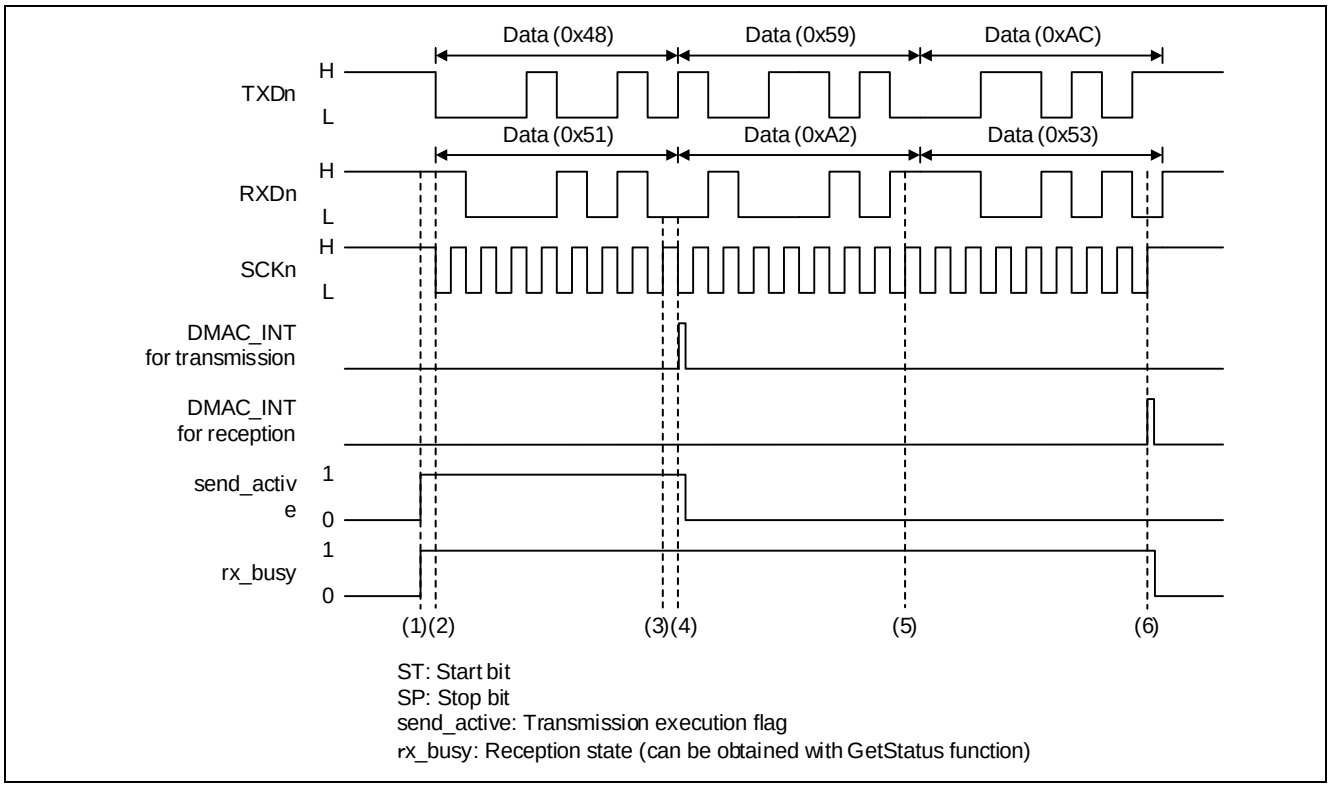


Figure 3-21 Transmission and Reception Operation using DMAC for Control (3 bytes received)

(1) When the Transfer function is executed, the tx_busy flag is set to "1" (transmission busy), the TXI and RXI interrupts are set as the DMAC transfer triggers, and then the first transmit data byte is written to the transmit data register (TDR).

(2) The second and the following transmit data bytes are transferred to the transmit data register (TDR) through DMA transfer.

(3) When data is received via the RXDn pin, the value in the receive data register (RDR) is transferred to the specified buffer through DMA transfer.

(4) When the specified number of bytes have been written, the DMAC transfer end interrupt on the transmitting side is generated. The send_active flag is cleared to "0" (transmission ready) in the interrupt handling process

(5) Each time one byte has been received, the RDR register value is transferred to the specified buffer through DMA transfer.

(6) After the specified size of data is transferred, the DMAC transfer end interrupt is generated. The rx_busy flag is cleared to "0" (reception wait state) in the interrupt handling process. If a callback function has been registered, the function is executed using ARM_USART_EVENT_TRANSFER_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the tx_busy and rx_busy flags are cleared to "0" (transmission wait state and reception wait state, respectively) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

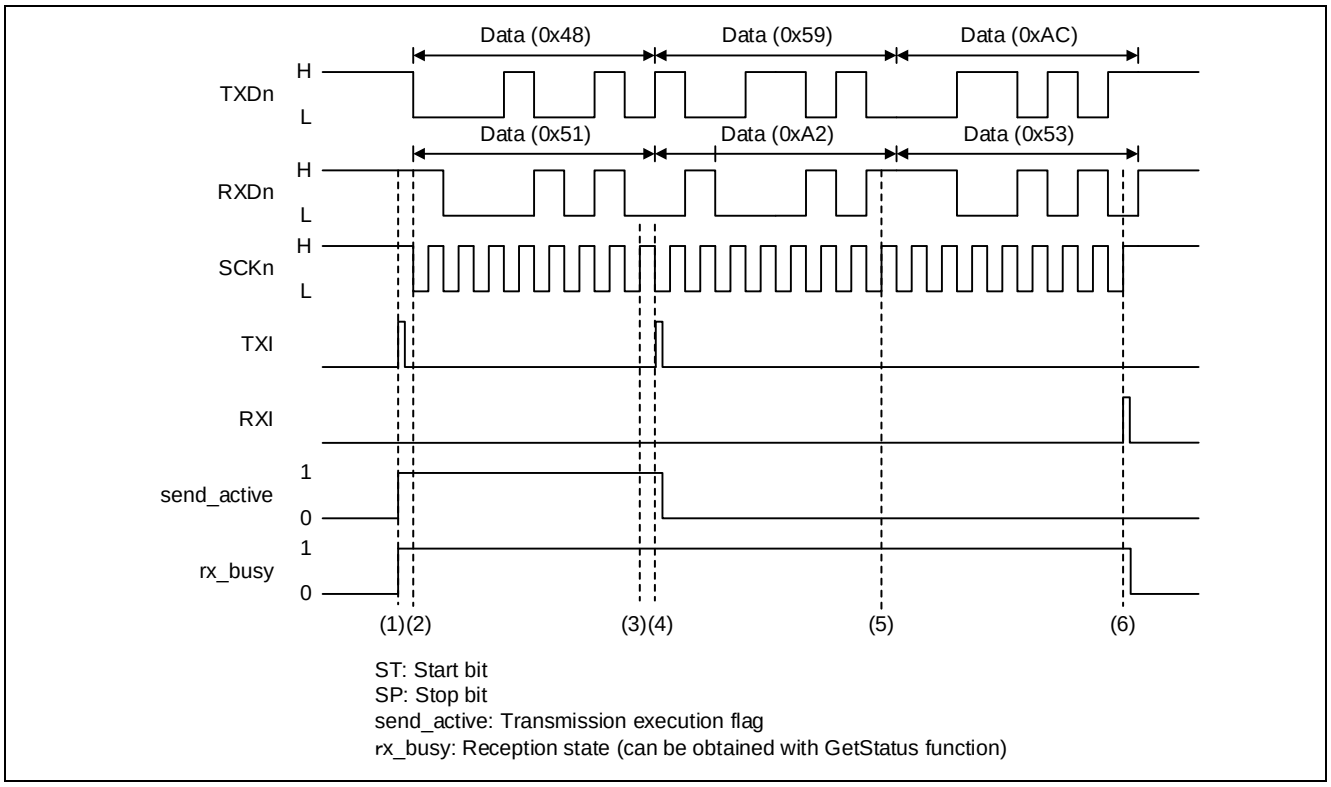


Figure 3-22 Transmission and Reception Operation using DTC for Control (3 bytes received)

(1) When the Transfer function is executed, the rx_busy flag is set to "1" (reception busy); the TXI interrupt is set as the DTC transfer trigger, and the TXI interrupt is enabled. Here, the TXI interrupt is generated and the first transmit data byte is written to the transmit data register (TDR).

(2) The second and the following transmit data bytes are transferred to the transmit data register (TDR) through DMA transfer.

(3) When data is received via the RXDn pin, the value in the receive data register (RDR) is transferred to the specified buffer through DMA transfer.

(4) When the specified number of bytes have been written, the TXI interrupt is generated. The send_active flag is cleared to "0" (transmission ready) in the interrupt handling process, and the TXI interrupt is disabled.

(5) Each time one byte has been received, the RDR register value is transferred to the specified buffer through DMA transfer.

(6) After the last data is transferred, the RXI interrupt is generated. The rx_busy flag is cleared to "0" (reception wait state) in the interrupt handling process. If a callback function has been registered, the function is executed using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

3.3 Clock Synchronous Slave Mode

3.3.1 Initial Setting Procedure of Clock Synchronous Slave Mode

Figure 3-23 shows the initial setting procedure of clock synchronous slave mode.

To enable transmission and reception, register the interrupts to use to NVIC in r_system_cfg.h. For details, see section 2.4 Communication Control and NVIC Interrupt Setting, Communication Control.

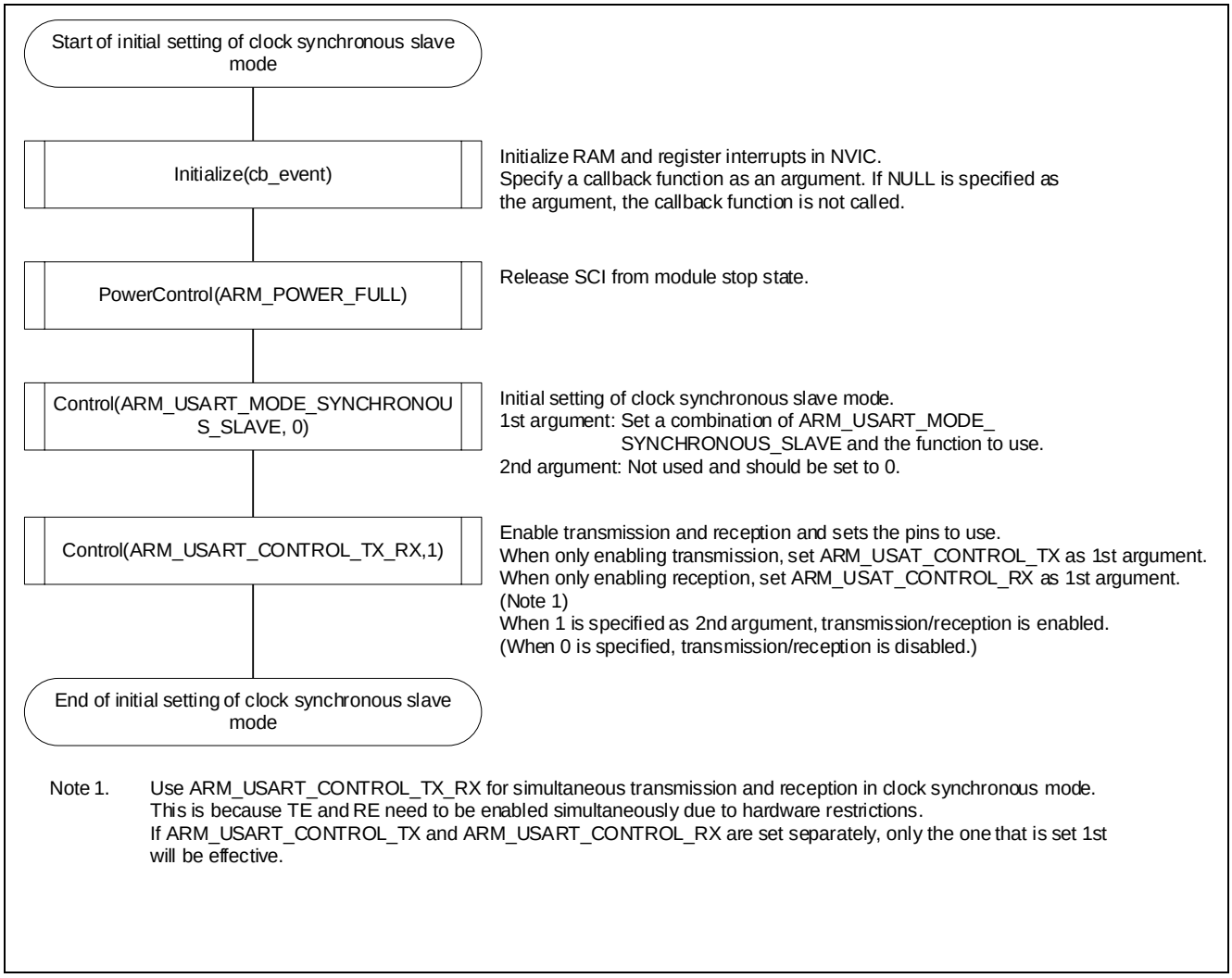


Figure 3-23 Clock Synchronous Slave Mode Initialization Procedure

3.3.2 Transmission in Clock Synchronous Slave Mode

Figure 3-24 shows the transmission procedure in clock synchronous slave mode

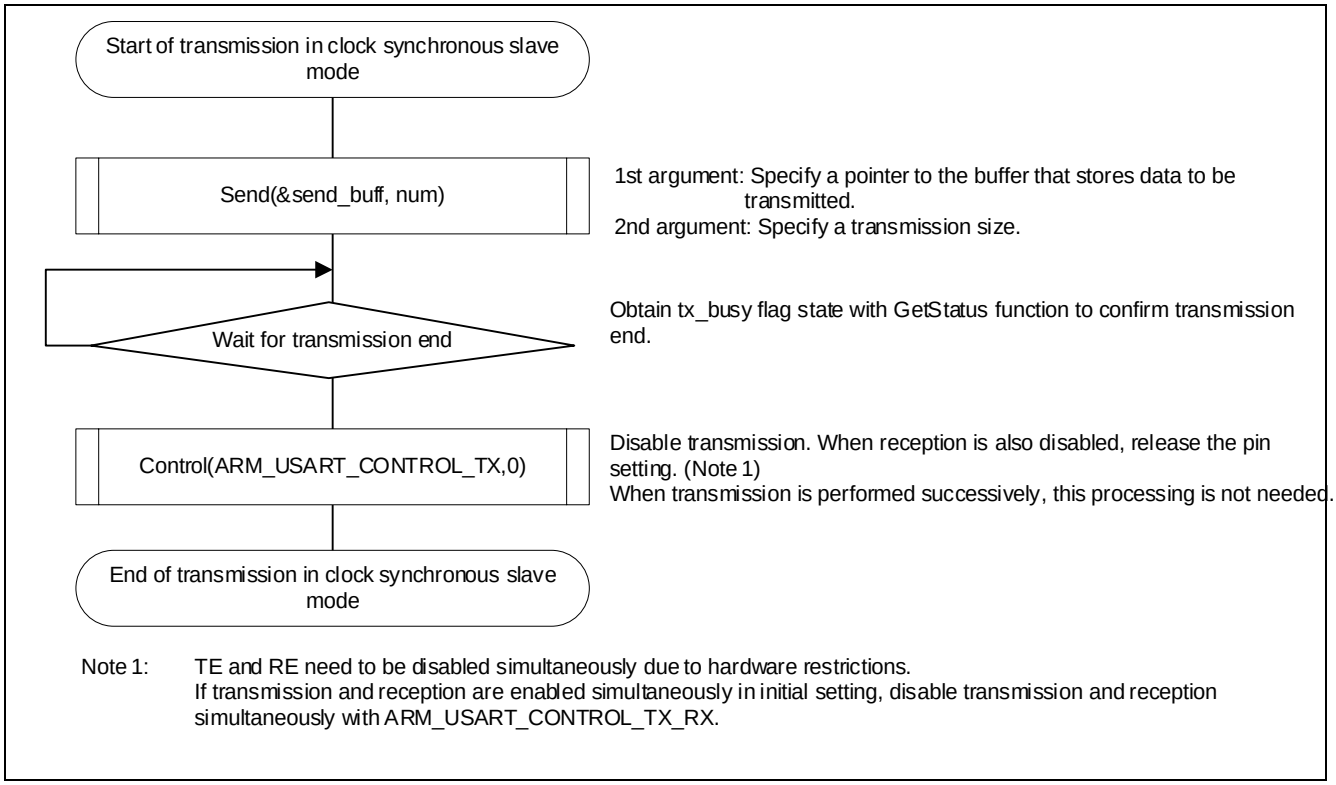


Figure 3-24 Transmission Procedure in Clock Synchronous Slave Mode

If a callback function has been set, the function is called upon completion of transmission using ARM_USART_EVENT_SEND_COMPLETE as an argument.

The specific transmission operation in clock synchronous slave mode is different depending on what method is used for communication control: the interrupt, DMAC, or DTC. Figure 3-25 shows the transmission operation when the interrupt is used for communication control, Figure 3-26 shows the transmission operation when the DMAC is used for communication control, and Figure 3-27 shows the transmission operation when the DTC is used for communication control.

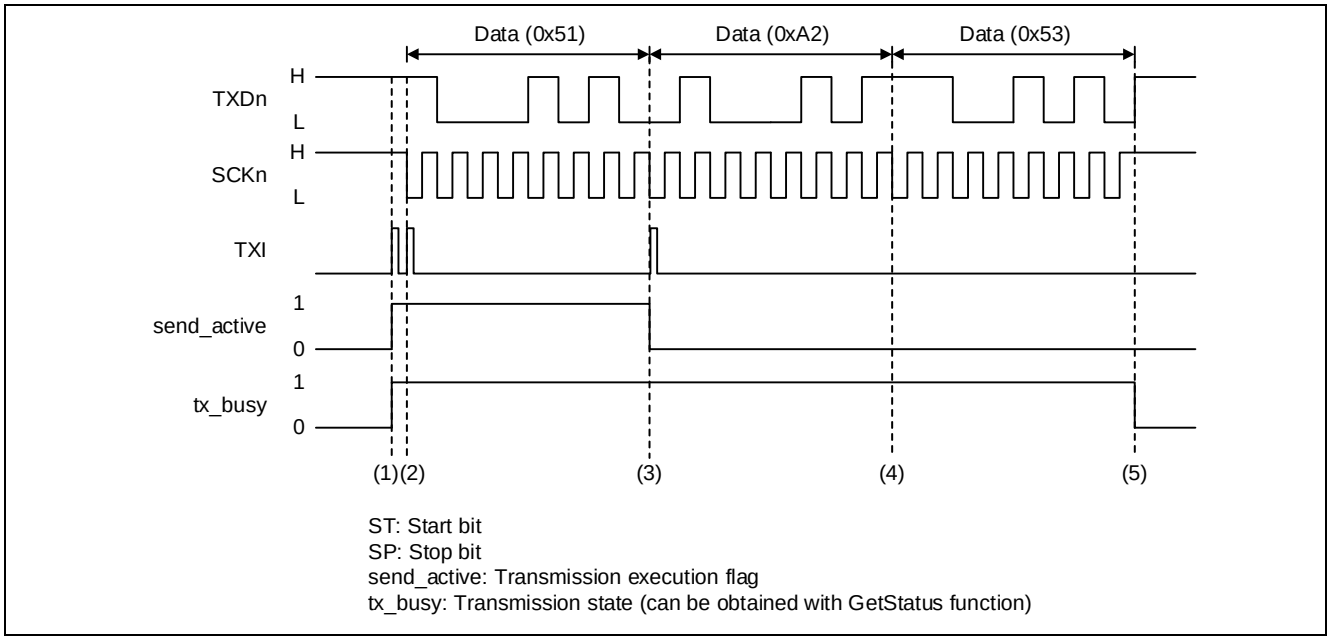


Figure 3-25 Transmission Operation using Interrupt for Control (3 bytes transmitted)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy). The TXI interrupt is also generated and the first data byte is written to the transmit data register (TDR).

(2) When the clock signal is input to the SCKn pin, the first data byte starts to be output from the TXD pin and the second TXI interrupt is generated. The second transmit data byte is written to the TDR register in the interrupt handling process.

(3) At the TXI interrupt after the last data is written, the TXI interrupt is disabled and the send_active flag is set to "0" (transmission ready). If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) After the second data byte is output, the last data, which has been written in step (3) is output. (Note)

(5) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

Note. If the Send function is executed while the tx_busy flag is "0" and the transmit buffer holds a value (period between (3) and (4)), the Send function is terminated after the tx_busy flag is set to "1". When the transmit buffer becomes empty (at (4)), the TXI interrupt is generated and the first data byte is written to the TDR register.

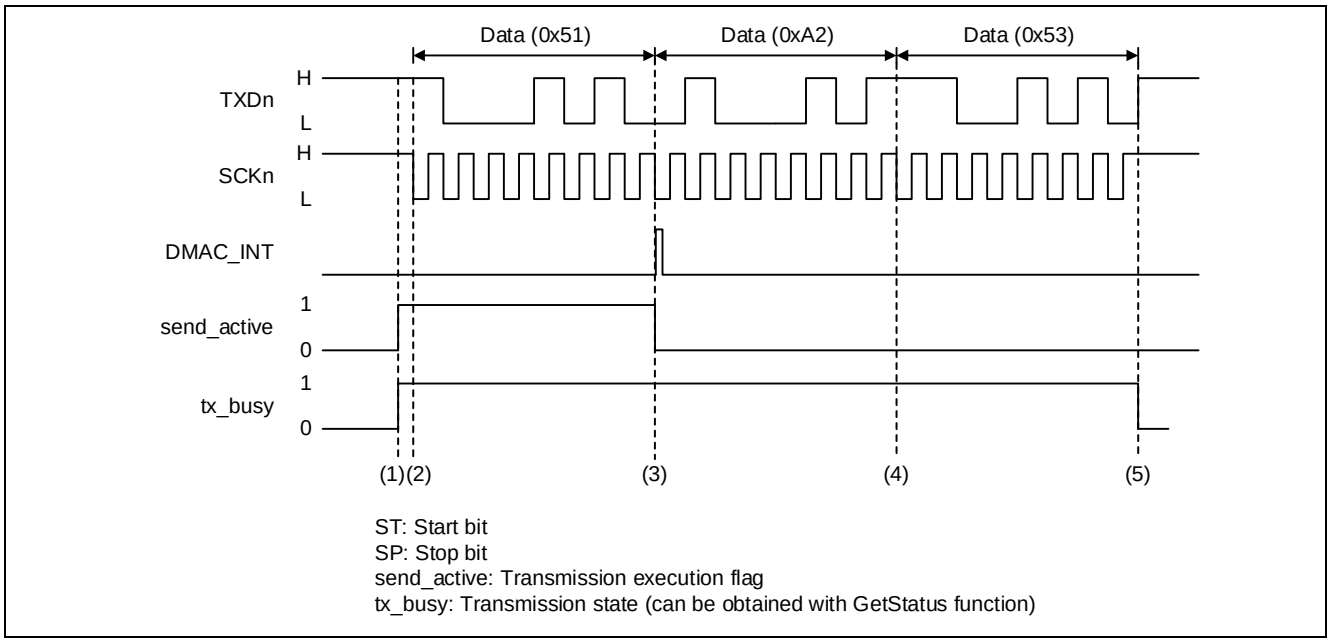


Figure 3-26 Transmission Operation using DMAC for Control (3 bytes transmitted)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy), the TXI interrupt is set as the DMAC transfer trigger, and the first data byte is written to the transmit data register (TDR).

(2) When the clock signal is input to the SCKn pin, the second and the following data bytes are transferred to the transmit data register (TDR) through DMA transfer.

(3) When the last data is transferred through DMA transfer, the DMAC transfer end interrupt is generated and the send_active flag is cleared to "0" (transmission ready). If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) After the second data byte is output, the last data, which has been written in step (3) is output. (Note)

(5) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

Note. If the Send function is executed while the tx_busy flag is "0" and the transmit buffer holds a value (period between (3) and (4)), execution is suspended until the transmit buffer becomes empty (at (4)), and step (1) is executed.
The suspension time can be changed with the value defined in SCI_CHECK_TDRE_TIMEOUT of r_usart_cfg.h. If the transmit buffer does not become empty within the specified time, the Send function returns ARM_DRIVER_ERROR_BUSY.

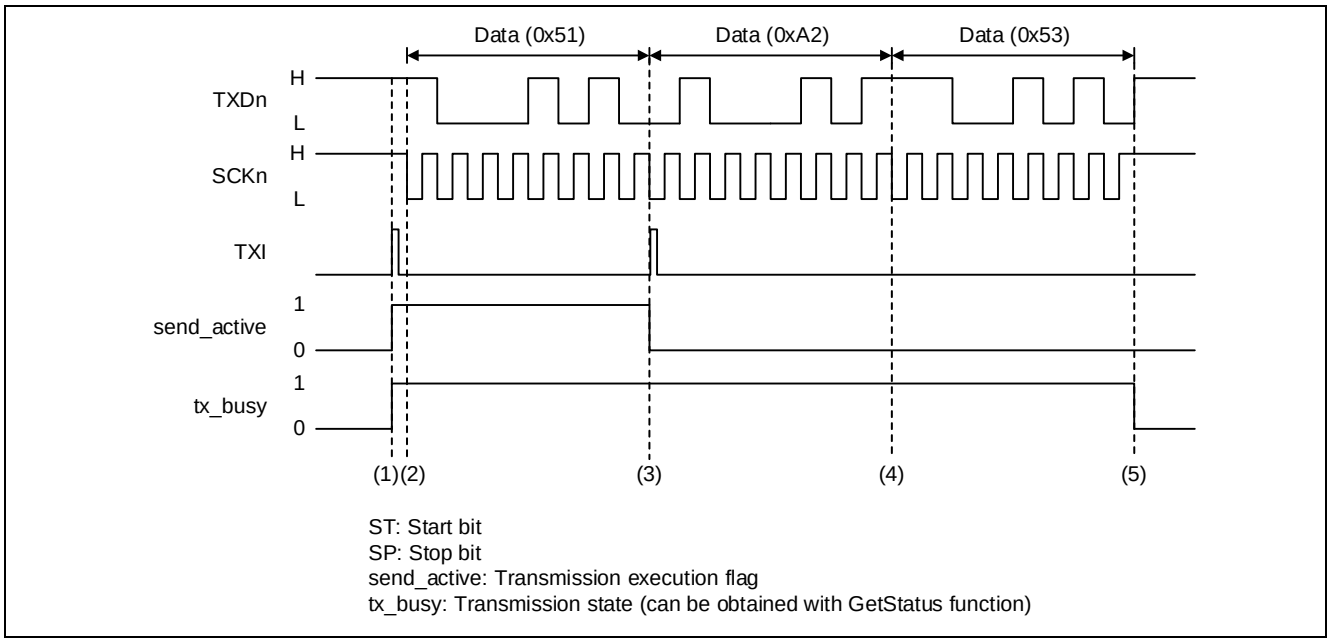


Figure 3-27 Transmission Operation using DTC for Control (3 bytes transmitted)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy), the TXI interrupt is set as the DMAC transfer trigger, and the TXI interrupt is enabled. Here, the TXI interrupt is generated and the first data byte is written to the transmit data register (TDR).

(2) When the clock signal is input to the SCKn pin, the second and the following data bytes are transferred to the TDR register through DMA transfer.

(3) When the last data is transferred through DMA transfer, the TXI interrupt is generated and the send_active flag is cleared to "0" (transmission ready). If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) After the second data byte is output, the last data, which has been written in step (3) is output. (Note)

(5) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

Note. If the Send function is executed while the tx_busy flag is "0" and the transmit buffer holds a value (period between (3) and (4)), execution is suspended until the transmit buffer becomes empty (at (4)), and step (1) is executed.
The suspension time can be changed with the value defined in SCI_CHECK_TDRE_TIMEOUT of r_usart_cfg.h. If the transmit buffer does not become empty within the specified time, the Send function returns ARM_DRIVER_ERROR_BUSY.

3.3.3 Reception in Clock Synchronous Slave Mode

Figure 3-28 shows the reception procedure in clock synchronous slave mode.

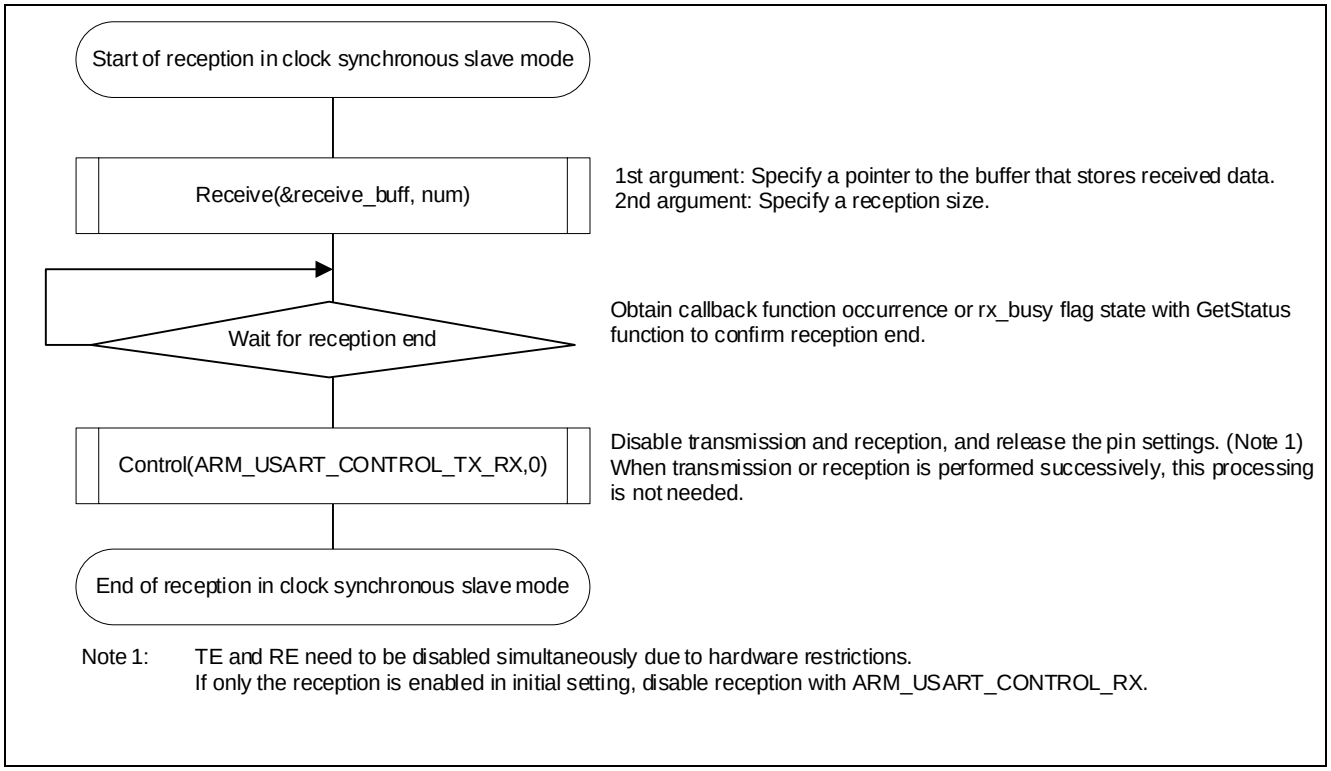


Figure 3-28 Reception Procedure in Clock Synchronous Slave Mode

If a callback function has been set, the function is called upon completion of reception using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument.

If a reception error is generated, the callback function is called using the error event information as an argument, and transmission and reception is terminated. Table 3-4 shows the error event information generated during reception in clock synchronous slave mode.

Table 3-4 Error Event Information Generated during Reception in Clock Synchronous Slave Mode

Error Event Information	Description
ARM_USART_EVENT_RX_OVERFLOW	An overrun error has been generated.

The specific reception operation in clock synchronous slave mode is different depending on what method is used for communication control: the interrupt, DMAC, or DTC. In addition, dummy data is written to the transmit buffer to output the clock signal if transmission is enabled. Dummy data to be output can be changed using the ARM_USART_SET_DEFAULT_TX_VALUE command.

Figure 3-29 shows the reception operation when the interrupt is used for communication control, Figure 3-30 shows the reception operation when the DMAC is used for communication control, andFigure 3-31 shows the reception operation when the DTC is used for communication control.

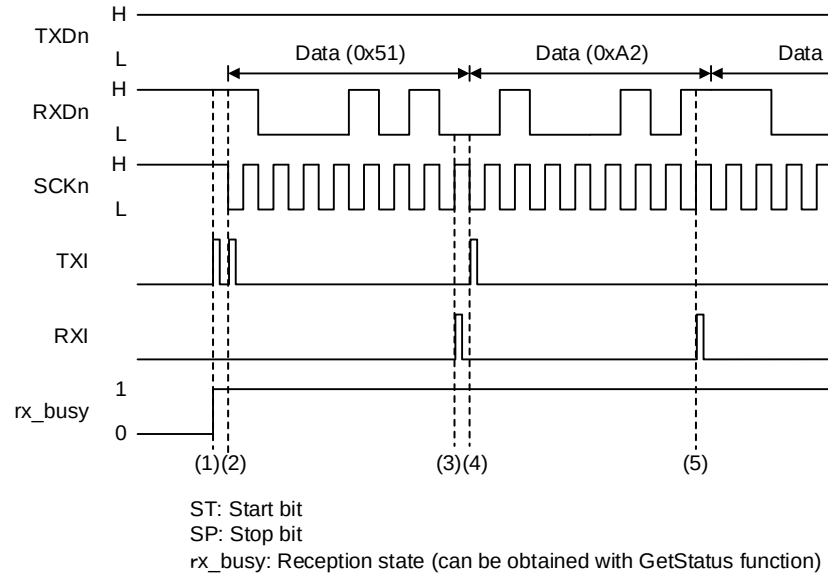


Figure 3-29 Reception Operation using Interrupt for Control (3 bytes received, transmission enabled, dummy data 0xFF)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy). The TXI interrupt is also generated and the dummy data is written to the transmit data register (TDR).

(2) When the clock signal is input to the SCKn pin, the dummy data starts to be output from the TXD pin, the second TXI interrupt is generated, and the second dummy data byte is written to the TDR register.

(3) When one byte has been received, the RXI interrupt is generated and the value is read from the receive data register (RDR) to the specified buffer.

(4) At the TXI interrupt after the specified number of bytes are written, the TXI interrupt is disabled.

(5) The RXI interrupt is generated each time one byte is received, and the received data is read from the RDR register.

(6) At the RXI interrupt after the last data is read, the rx_busy flag is cleared to "0" (reception wait state). If a callback function has been registered, the function is executed using ARM_USART_EVENT_ RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

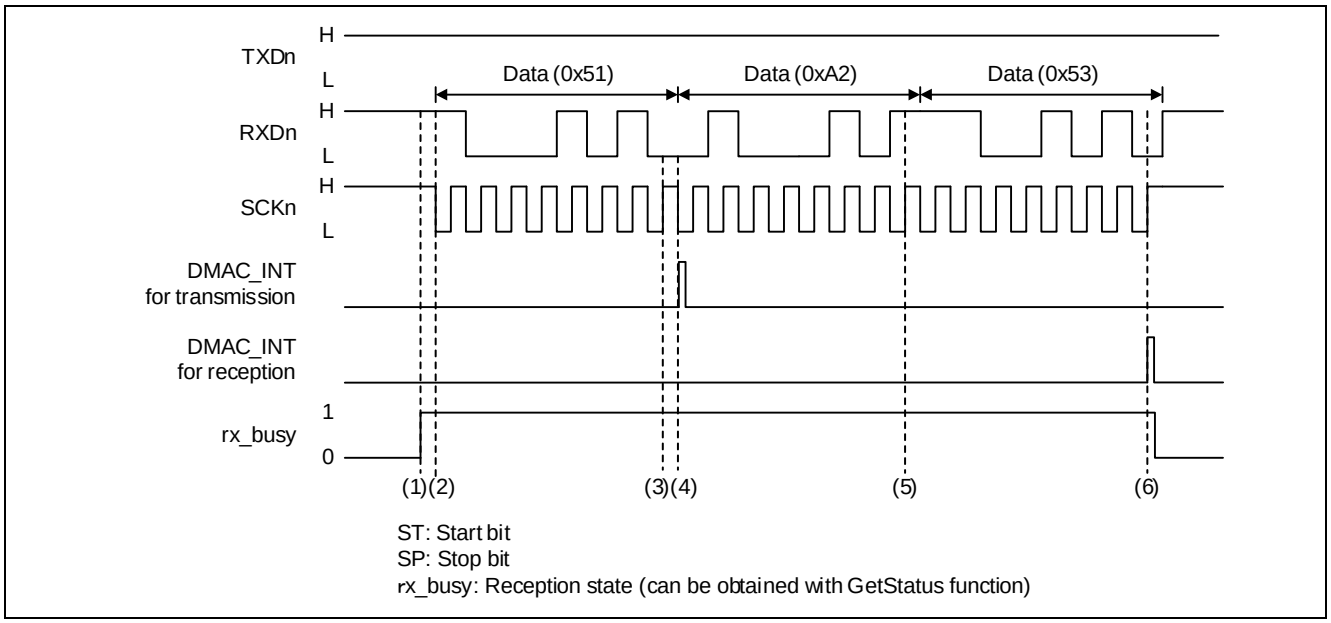


Figure 3-30 Reception Operation using DMAC for Control (3 bytes received, transmission enabled, dummy data 0xFF)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy), the TXI and RXI interrupts are set as the DMAC transfer triggers, and then the first dummy data is written to the transmit data register (TDR).

(2) When the clock signal is input to the SCKn pin, the dummy data starts to be output from the TXD pin and the second and the following dummy data are transferred to the transmit data register (TDR).

(3) When data is received via the RXDn pin, the value in the receive data register (RDR) is transferred to the specified buffer through DMA transfer.

(4) When the specified number of bytes have been written, the DMAC transfer end interrupt on the transmitting side is generated.

(5) Each time one byte has been received, the RDR register value is transferred to the specified buffer through DMA transfer.

(6) After the specified size of data is transferred, the DMAC transfer end interrupt is generated. The rx_busy flag is cleared to "0" (reception wait state) in the interrupt handling process. If a callback function has been registered, the function is executed using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

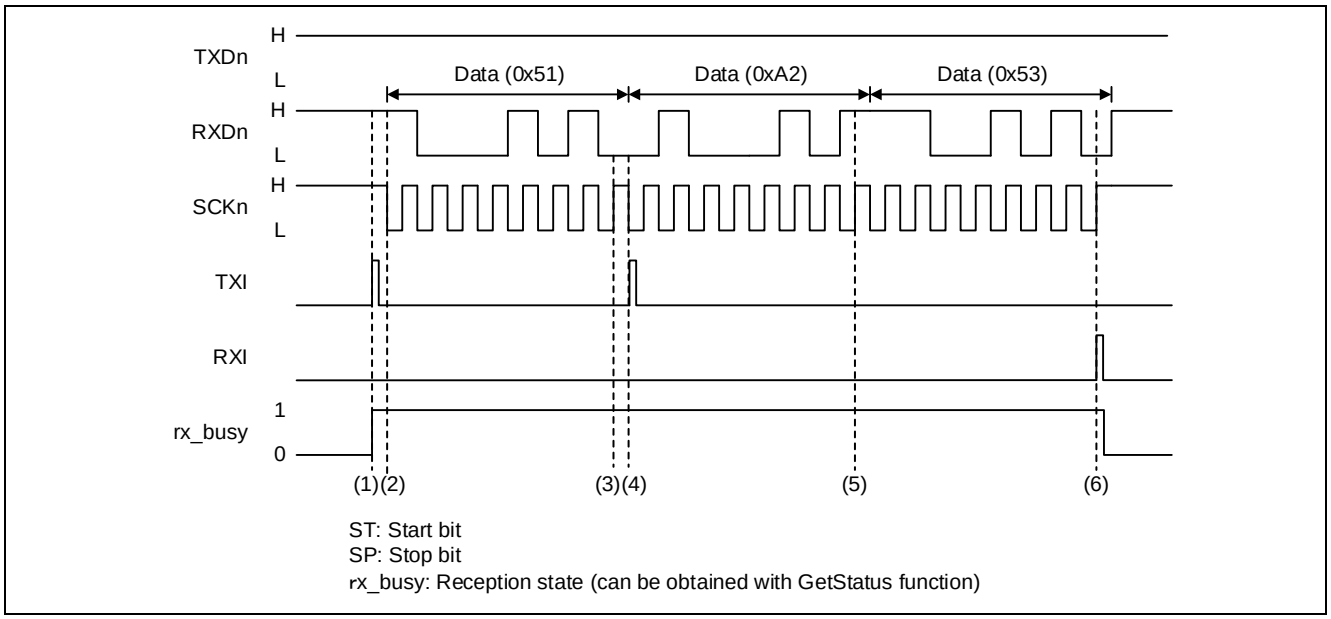


Figure 3-31 Reception Operation using DTC for Control (3 bytes received, transmission enabled, dummy data 0xFF)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy), the TXI interrupt is set as the DTC transfer trigger, and the TXI interrupt is enabled. Here, the TXI interrupt is generated and the first dummy data is written to the transmit data register (TDR)

(2) When the clock signal is input to the SCKn pin, the first data byte starts to be output from the TXD pin and the second and the following dummy data bytes are transferred to the transmit data register (TDR) through DMA transfer.

(3) When data is received via the RXDn pin, the value in the receive data register (RDR) is transferred to the specified buffer through DMA transfer.

(4) When the specified number of bytes have been written, the TXI interrupt is generated.

(5) Each time one byte has been received, the RDR register value is transferred to the specified buffer through DMA transfer.

(6) After the last data is transferred, the RXI interrupt is generated. The rx_busy flag is cleared to "0" (reception wait state) in the interrupt handling process. If a callback function has been registered, the function is executed using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

3.3.4 Transmission and Reception in Clock Synchronous Slave Mode

Figure 3-32shows the transmission and reception procedure in clock synchronous slave mode.

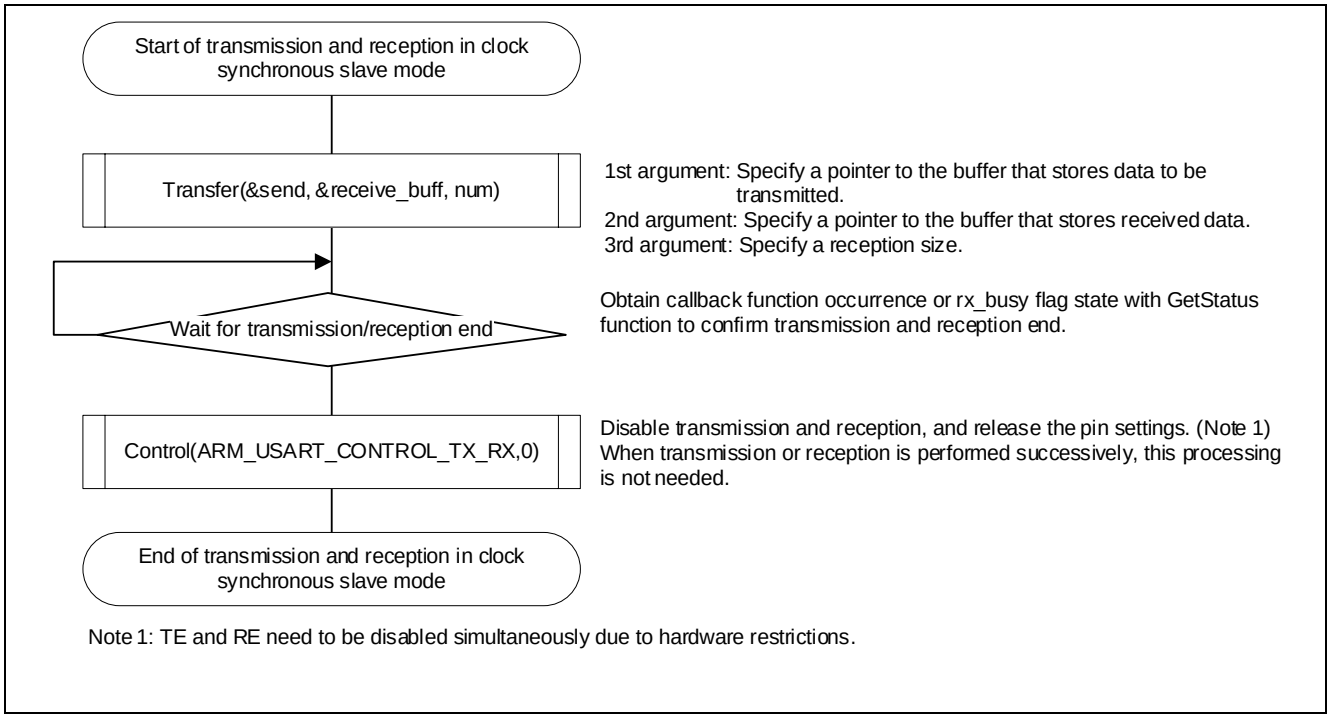


Figure 3-32 Transmission and Reception Procedure in Clock Synchronous Slave Mode

If a callback function has been set, the function is called upon completion of reception using ARM_USART_EVENT_TRANSFER_COMPLETE as an argument.

If a reception error is generated, the callback function is called using the error event information as an argument, and transmission and reception are terminated. Table 3-5 shows the error event information generated during transmission and reception in clock synchronous slave mode.

Table 3-5 Error Event Information Generated during Transmission and Reception in Clock Synchronous Slave Mode

Error Event Information	Description
ARM_USART_EVENT_RX_OVERFLOW	An overrun error has been generated.

The specific transmission and reception operation in clock synchronous slave mode is different depending on what method is used for communication control: the interrupt, DMAC, or DTC. Figure 3-33 shows the operation when the interrupt is used for communication control, Figure 3-34 shows the operation when the DMAC is used for communication control, andFigure 3-35 shows the operation when the DTC is used for communication control.

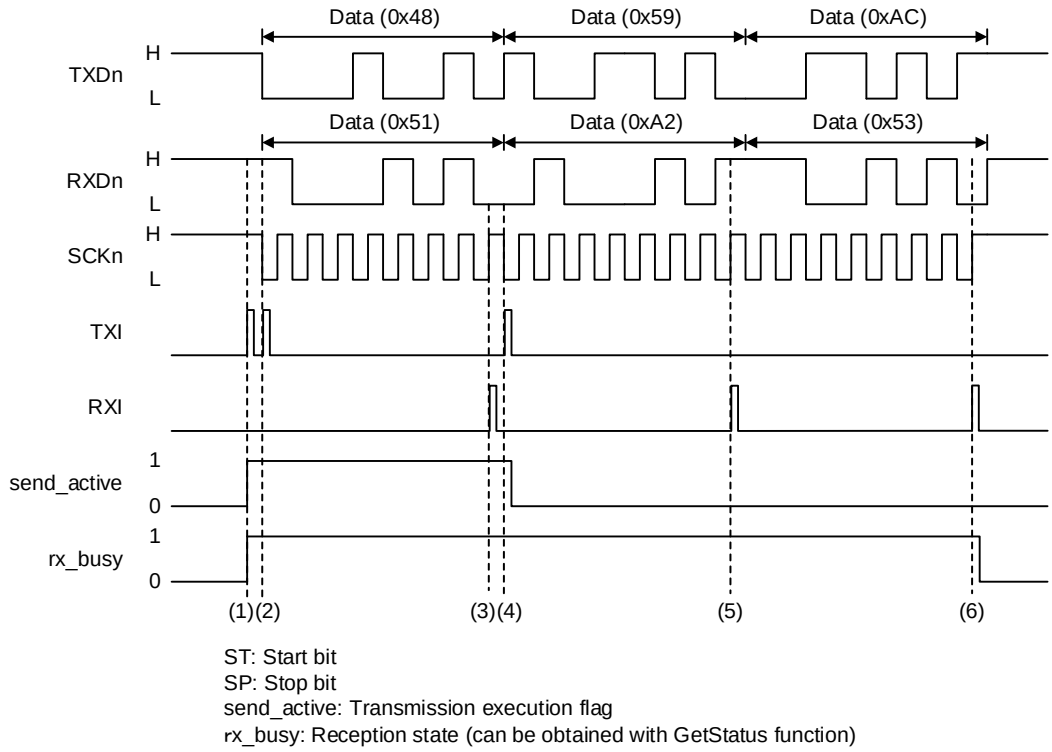


Figure 3-33 Transmission and Reception Operation using Interrupt for Control (3 bytes received)

(1) When the Transfer function is executed, the tx_busy flag is set to "1" (transmission busy), the rx_busy flag is set to "1" (reception busy), and the TXI interrupt is enabled. Here, the TXI interrupt is generated and the first transmit data byte is written to the transmit data register (TDR).

(2) At the second TXI interrupt, the second transmit data byte is written to the TDR register.

(3) When one byte has been received, the RXI interrupt is generated and the value is read from the receive data register (RDR) to the specified buffer.

(4) At the TXI interrupt after the specified number of bytes are written, the send_active flag is cleared to "0" (transmission ready), and the TXI interrupt is disabled.

(5) The RXI interrupt is generated each time one byte is received, and the received data is read from the RDR register.

(6) At the RXI interrupt after the last data is read, the rx_busy flag is cleared to "0" (reception wait state). If a callback function has been registered, the function is executed using ARM_USART_EVENT_TRANSFER_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the tx_busy flag is cleared to "0" (transmission wait state) and rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

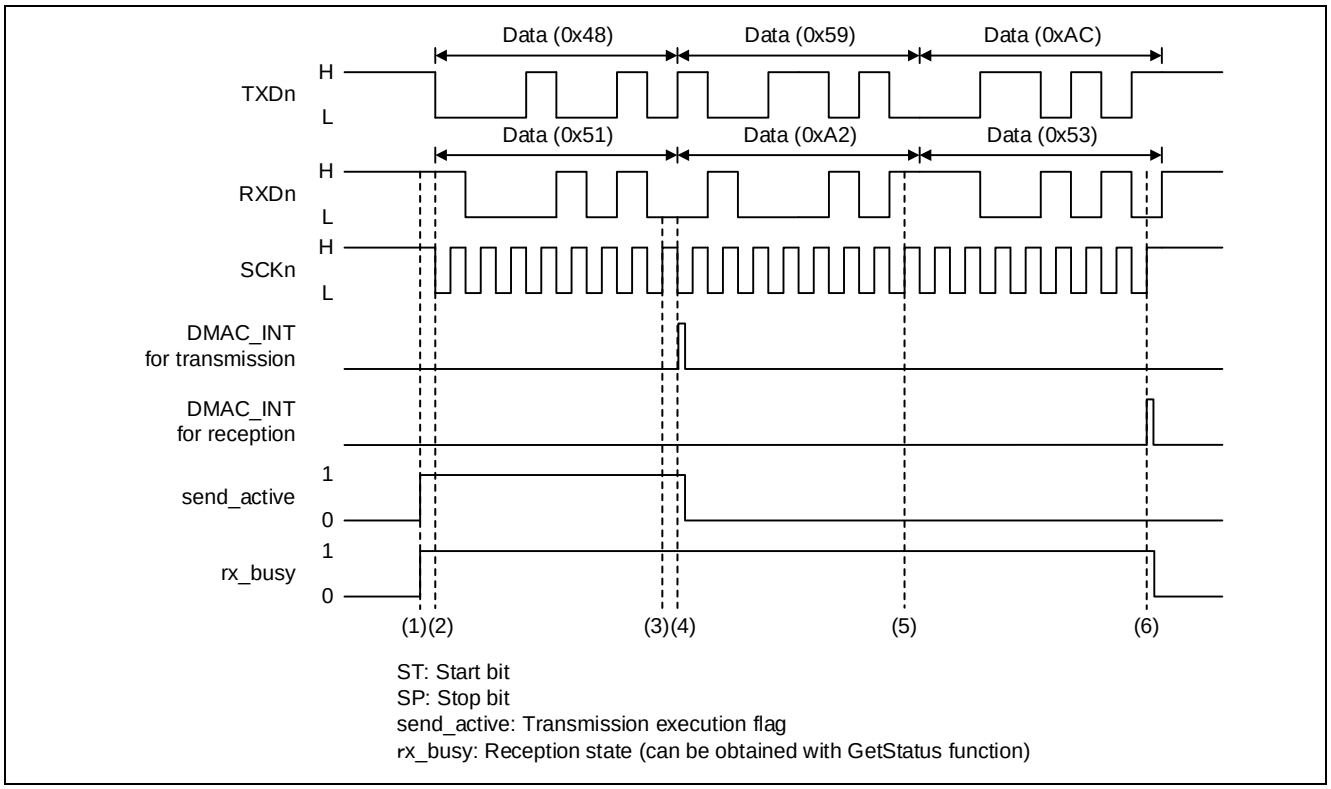


Figure 3-34 Transmission and Reception Operation using DMAC for Control (3 bytes received)

(1) When the Transfer function is executed, the tx_busy flag is set to "1" (transmission busy) and the rx_busy flag is set to "1" (reception busy); the TXI and RXI interrupts are set as the DMAC transfer triggers, and the first data byte is written to the transmit data register (TDR).

(2) The second and the following transmit data bytes are transferred to the transmit data register (TDR) through DMA transfer.

(3) When data is received via the RXDn pin, the value in the receive data register (RDR) is transferred to the specified buffer through DMA transfer.

(4) When the specified number of bytes have been written, the DMAC transfer end interrupt on the transmitting side is generated. The send_active flag is cleared to "0" (transmission ready) in the interrupt handling process.

(5) Each time one byte has been received, the RDR register value is transferred to the specified buffer through DMA transfer.

(6) After the specified size of data is transferred, the DMAC transfer end interrupt is generated on the receiving side. The rx_busy flag is cleared to "0" (reception wait state) in the interrupt handling process. If a callback function has been registered, the function is executed using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the tx_busy flag is cleared to "0" (transmission wait state) and rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

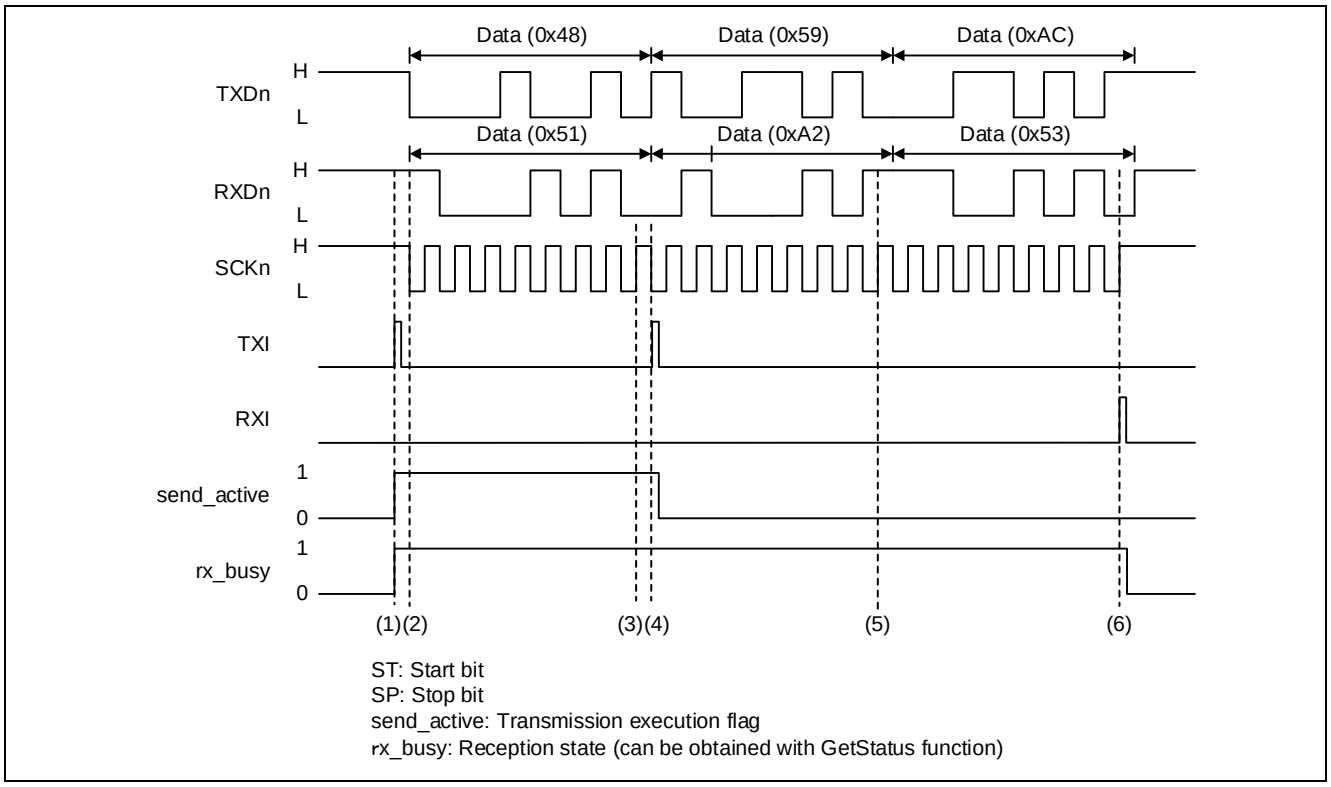


Figure 3-35 Transmission and Reception Operation using DTC for Control (3 bytes received)

(1) When the Transfer function is executed, the rx_busy flag is set to "1" (reception busy), the TXI interrupt is set as the DTC transfer trigger, and the TXI interrupt is enabled. Here, the TXI interrupt is generated and the first transmit data is written to the transmit data register (TDR)

(2) The second and the following transmit data bytes are transferred to the transmit data register (TDR) through DMA transfer.

(3) When data is received via the RXDn pin, the value in the receive data register (RDR) is transferred to the specified buffer through DMA transfer.

(4) When the specified number of bytes have been written, the TXI interrupt is generated. In the interrupt handling process, the send_active flag is cleared to "0" (transmission ready).

(5) Each time one byte has been received, the RDR register value is transferred to the specified buffer through DMA transfer.

(6) After the last data is transferred, the RXI interrupt is generated. The rx_busy flag is cleared to "0" (reception wait state) in the interrupt handling process. If a callback function has been registered, the function is executed using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument. (Note)

Note. If the reception error is generated, the ERI interrupt is generated, and the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

3.4 Smart Card Mode

3.4.1 Initial Setting Procedure of Smart Card Mode

Figure 3-36 shows the initial setting procedure of smart card mode.

To enable transmission and reception, register the interrupts to use to NVIC in r_system_cfg.h. For details, see section 2.4 Communication Control and NVIC Interrupt Setting, Communication Control.

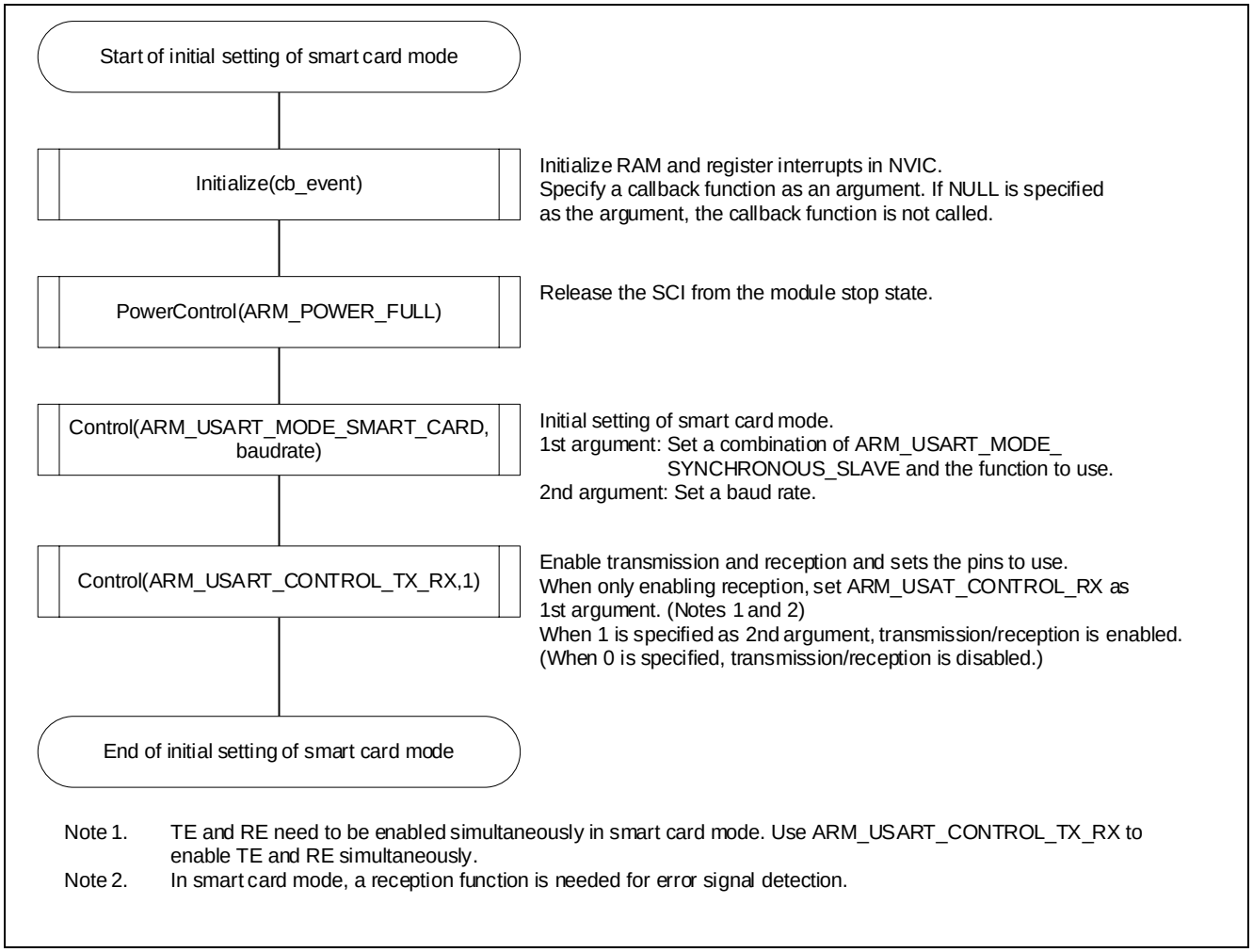


Figure 3-36 Smart Card Mode Initialization Procedure

3.4.2 Transmission in Smart Card Mode

Figure 3-37 shows the transmission procedure in smart card mode.

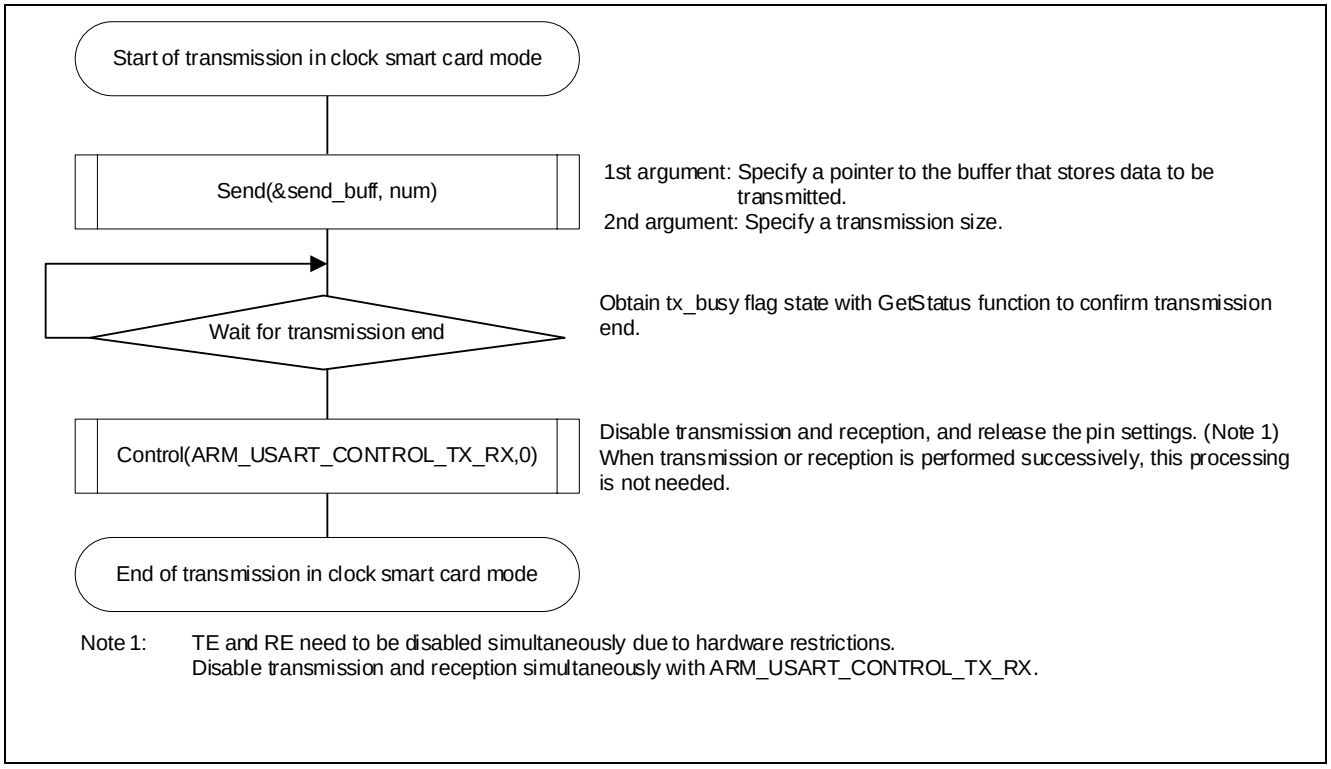


Figure 3-37 Transmission Procedure in Smart Card Mode

If a callback function has been set, the function is called upon completion of transmission using ARM_USART_EVENT_SEND_COMPLETE as an argument.

The specific transmission operation in smart card mode is different depending on what method is used for communication control: the interrupt, DMAC, or DTC. Figure 3-38 shows the transmission operation when the interrupt is used for communication control, Figure 3-39 shows the transmission operation when the DMAC is used for communication control, and Figure 3-40 shows the transmission operation when the DTC is used for communication control.

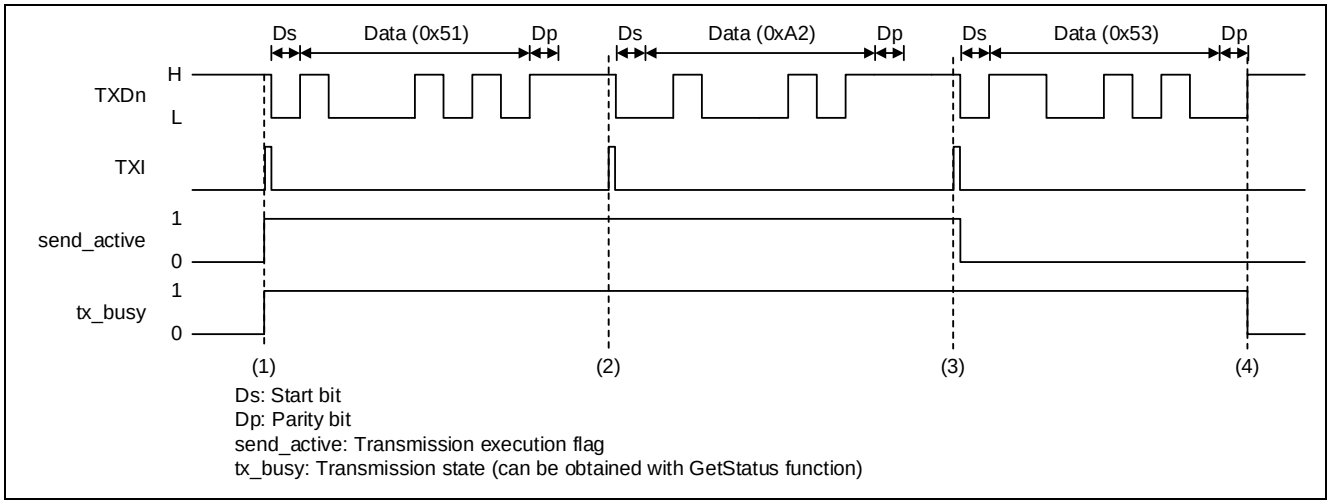


Figure 3-38 Transmission Operation using Interrupt for Control (3 bytes transmitted, even parity)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy). The TXI interrupt is also generated and the first data byte is written to the transmit data register (TDR).

(2) After the first data byte is output, the second TXI interrupt is generated if the error signal has not been received from the receiving station. The second transmit data byte is written to the TDR register in the interrupt handling process.

(3) At the TXI interrupt after the last data is written, the TXI interrupt is disabled and the send_active flag is cleared to "0" (transmission ready). If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

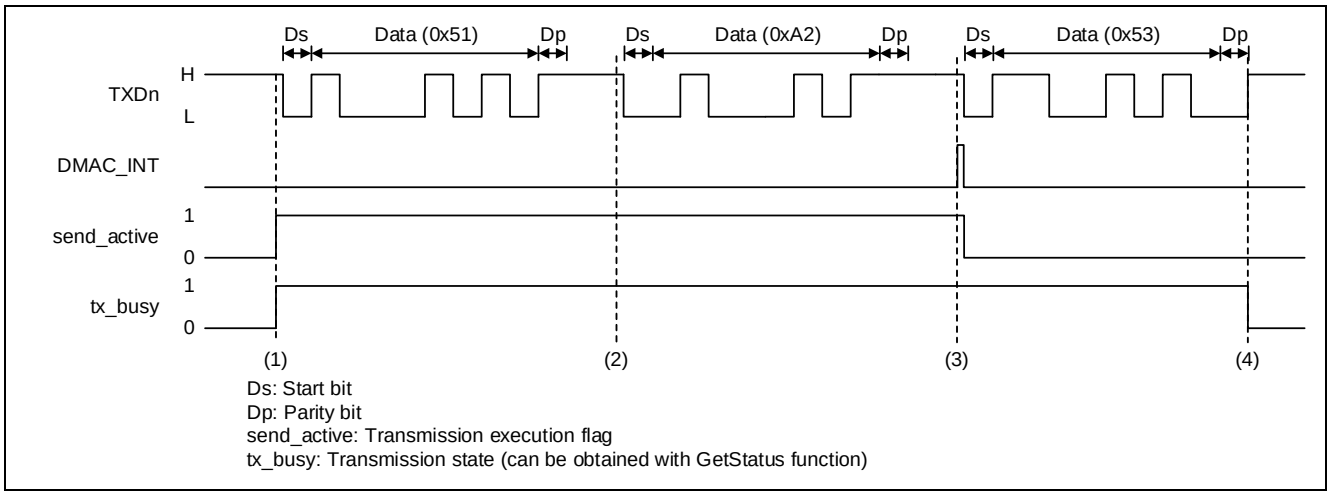


Figure 3-39 Transmission Operation using DMAC for Control (3 bytes transmitted, even parity)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy), the TXI interrupt is set as the DMAC transfer trigger, and the first data byte is written to the transmit data register (TDR).

(2) After the first data byte is output, if the error signal has not been received from the receiving station, the second and the following transmit data bytes are transferred to the transmit data register (TDR) through DMA transfer.

(3) When the last data is transferred through DMA transfer, the DMAC transfer end interrupt is generated and the send_active flag is cleared to "0" (transmission ready). If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

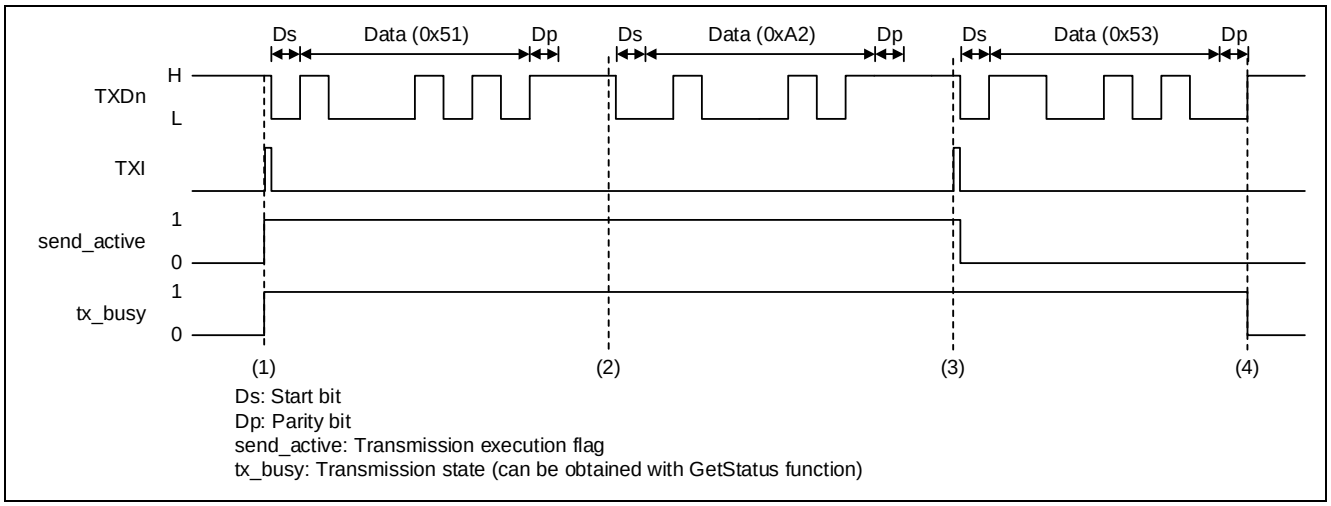


Figure 3-40 Transmission Operation using DTC for Control (3 bytes transmitted, even parity)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy), the TXI interrupt is set as the DTC transfer trigger, and the TXI interrupt is enabled. Here, the TXI interrupt is generated and the first data byte is written to the transmit data register (TDR).

(2) After the first data byte is output, if the error signal has not been received from the receiving station, the second and the following transmit data bytes are transferred to the transmit data register (TDR) through DMA transfer.

(3) When the last data is transferred through DMA transfer, the TXI interrupt is generated and the send_active flag is cleared to "0" (transmission ready). If a callback function has been registered, the function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument.

(4) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

3.4.3 Error Signal Reception during Transmission in Smart Card Mode

If the error signal is received during transmission in smart card mode, the same data is automatically re-transmitted by hardware. In addition, the ERI interrupt is generated and the callback function is called using ARM_USART_EVENT_RX_PARITY_ERROR as an argument if the callback function has been registered.

Figure 3-42 shows the operation when the error signal is received during transmission.

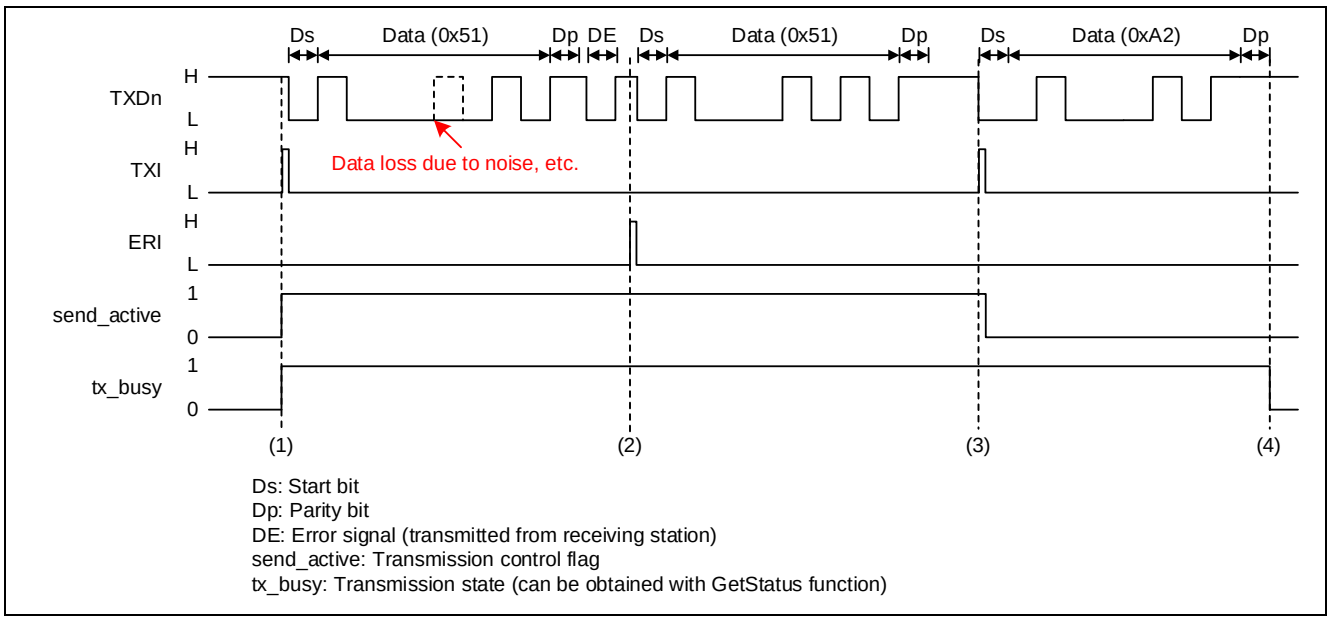


Figure 3-41 Operation When Error Signal is Received (2 bytes transmitted, even parity)

(1) When the Send function is executed, the tx_busy flag is set to "1" (transmission busy). The TXI interrupt is also generated and the first data byte is written to the transmit data register (TDR).

(2) If the error signal from the receiving station is received after the first byte data is output, the ERI interrupt is generated. In the interrupt handling process, the callback function is executed using ARM_USART_EVENT_RX_PARITY_ERROR as an argument if a callback function has been registered. The first byte data is also re-transmitted automatically.

(3) If the error signal from the receiving station is not received after the re-transmitted data is output, the second TXI interrupt is generated. In the interrupt handling process, second byte data is written, the TXI interrupt is disabled, and the send_active flag is cleared to "0" (transmission ready). In addition, the callback function is executed using ARM_USART_EVENT_SEND_COMPLETE as an argument if a callback function has been registered. (Note)

(4) When transmission is completed, the tx_busy flag is cleared to "0" (transmission end).

Note. If the error signal is received after the last data is output, the ERI interrupt is generated and data is re-transmitted as well like in step (2).

3.4.4 Reception in Smart Card Mode

Figure 3-42 shows the reception procedure in smart card mode.

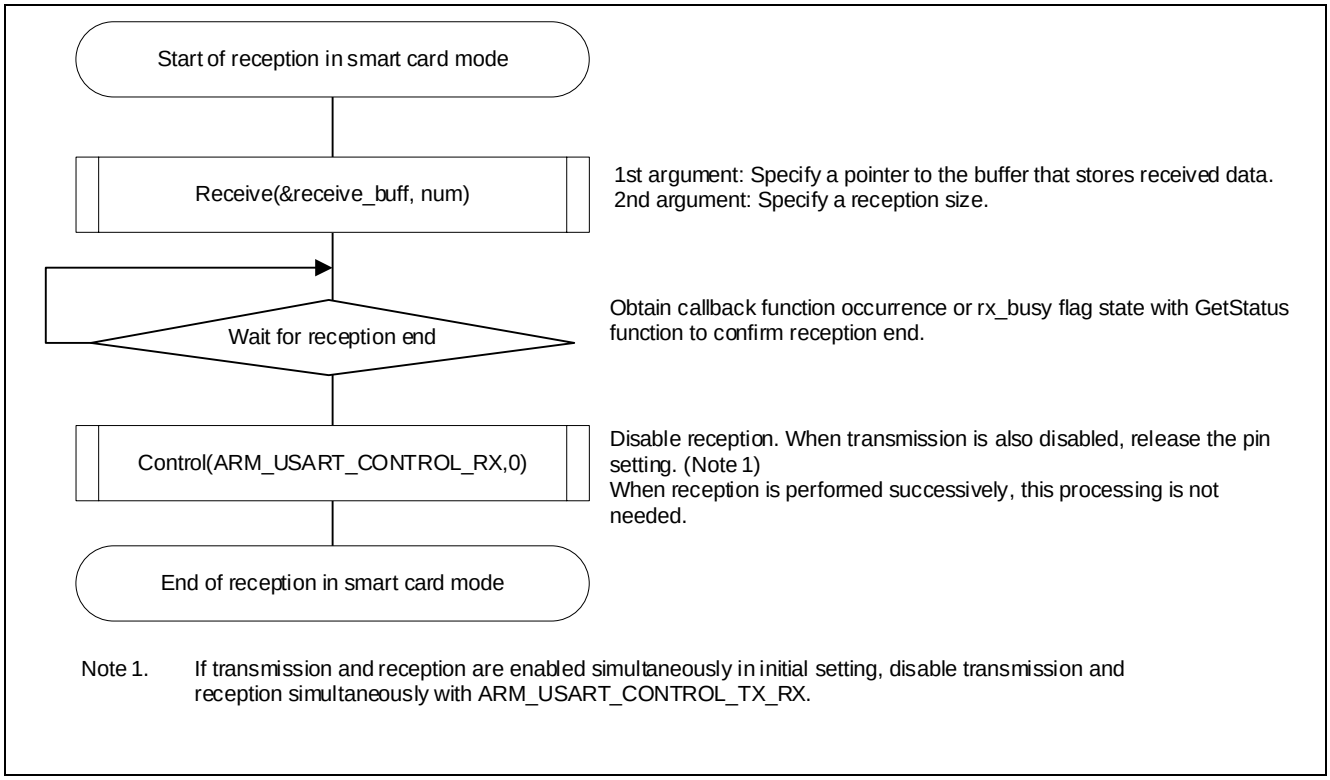


Figure 3-42 Reception Procedure in Smart Card Mode

If a callback function has been set, the function is called upon completion of reception using ARM_USART_EVENT_RECEIVE_COMPLETE as an argument.

In addition, the callback function is called using the error event information as an argument if the reception error is generated. If the reception error is an overrun error, transmission and reception are terminated. If it is a parity error, the error signal is output and reception control is continued. Table 3-6 shows the error event information generated during reception in smart card mode.

Table 3-6 Error Event Information Generated during Reception in Smart Card Mode

Error Event Information (Note)	Description
ARM_USART_EVENT_RX_OVERFLOW	An overrun error has been generated.
ARM_USART_EVENT_RX_PARITY_ERROR	A parity error has been generated.

Note: If more than one error is generated, a callback function is called using the ORed error event information as an argument.

The specific reception operation in smart card mode is different depending on what method is used for communication control: the interrupt, DMAC, or DTC. Figure 3-43 shows the reception operation when the interrupt is used for communication control, Figure 3-44 shows the reception operation when the DMAC is used for communication control, and Figure 3-45 shows the reception operation when the DTC is used for communication control.

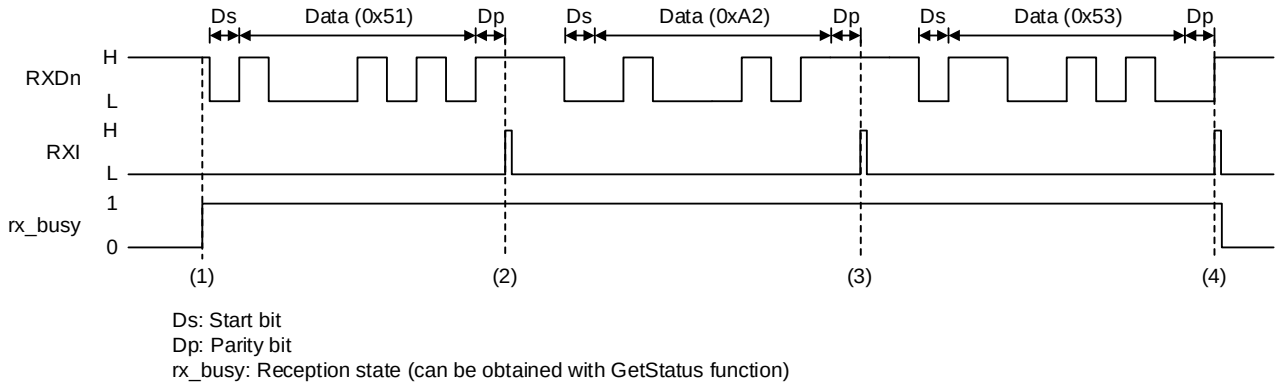


Figure 3-43 Reception Operation using Interrupt for Control (3 bytes received, even parity)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy).

(2) When one byte has been received, the RXI interrupt is generated and the value is read from the receive data register (RDR) to the specified buffer.

(3) The RXI interrupt is generated each time one byte is received, and the received data is read from the RDR register.

(4) At the RXI interrupt after the last data is read, the rx_busy flag is cleared to "0" (reception wait state). If a callback function has been registered, the function is executed using ARM_USART_EVENT_ RECEIVE_COMPLETE as an argument. (Note)

Note. If a reception error is generated, the ERI interrupt is generated. The generated error is identified in the interrupt handling process. If the reception error is a parity error, the error signal is output. If the reception error is an overrun error, the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

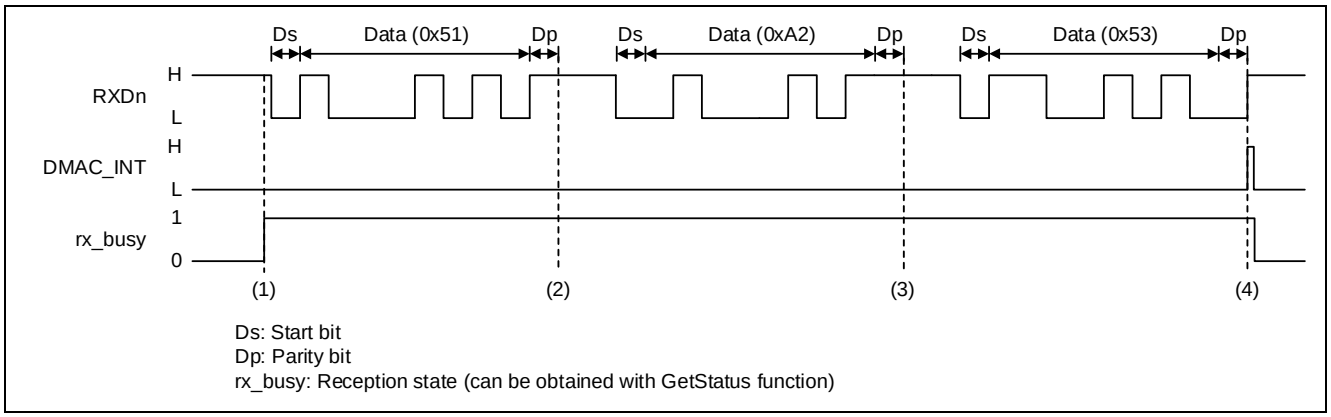


Figure 3-44 Reception Operation using DMAC for Control (3 bytes received, even parity)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy).

(2) When one byte has been received, the value in the receive data register (RDR) is transferred to the specified buffer through DMA transfer.

(3) DMA transfer is generated each time one byte is received, and the received data is read from the RDR register.

(4) At the DMAC transfer end interrupt after the last data is transferred, the rx_busy flag is cleared to "0" (reception wait state). If a callback function has been registered, the function is executed using ARM_USART_EVENT_ RECEIVE_COMPLETE as an argument. (Note)

Note. If a reception error is generated, the ERI interrupt is generated. The generated error is identified in the interrupt handling process. If the reception error is a parity error, the error signal is output. If the reception error is an overrun error, the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

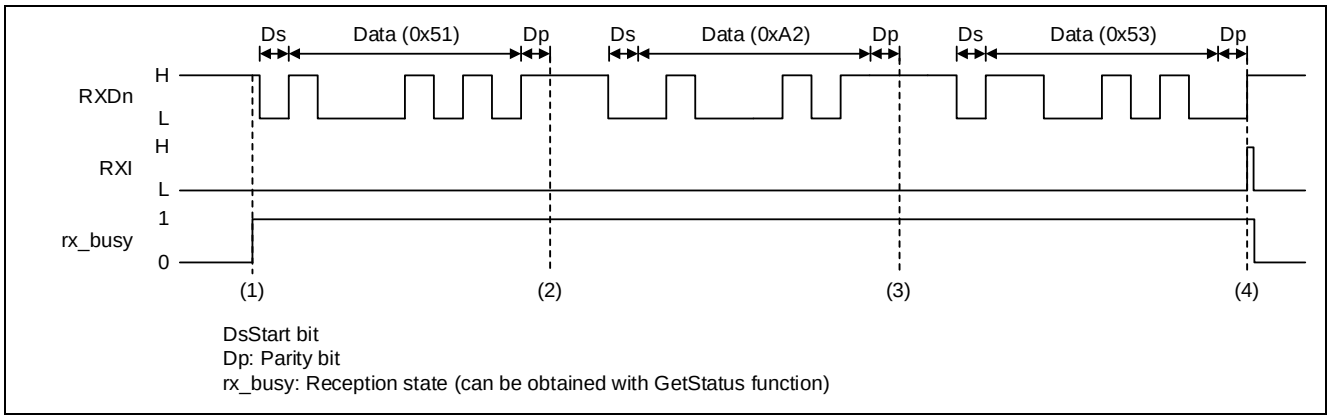


Figure 3-45 Reception Operation using DTC for Control (3 bytes received, even parity)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy).

(2) When one byte has been received, the value in the receive data register (RDR) is transferred to the specified buffer through DMA transfer.

(3) DMA transfer is generated each time one byte is received, and the received data is read from the RDR register.

(4) At the RXI interrupt after the last data is transferred, the rx_busy flag is cleared to "0" (reception wait state). If a callback function has been registered, the function is executed using ARM_USART_EVENT_ RECEIVE_COMPLETE as an argument. (Note)

Note. If a reception error is generated, the ERI interrupt is generated. The generated error is identified in the interrupt handling process. If the reception error is a parity error, the error signal is output. If the reception error is an overrun error, the rx_busy flag is cleared to "0" (reception wait state) to clear the error state. If a callback function has been registered, the function is executed using the error event information as an argument.

3.4.5 Error Signal Transmission during Reception in Smart Card Mode

If a parity error is detected during reception in smart card mode, the error signal is transmitted automatically by hardware. In addition, the ERI interrupt is generated and the callback function is called using ARM_USART_EVENT_RX_PARITY_ERROR as an argument if the callback function has been registered.

Figure 3-46 shows the operation when the parity error is detected during transmission.

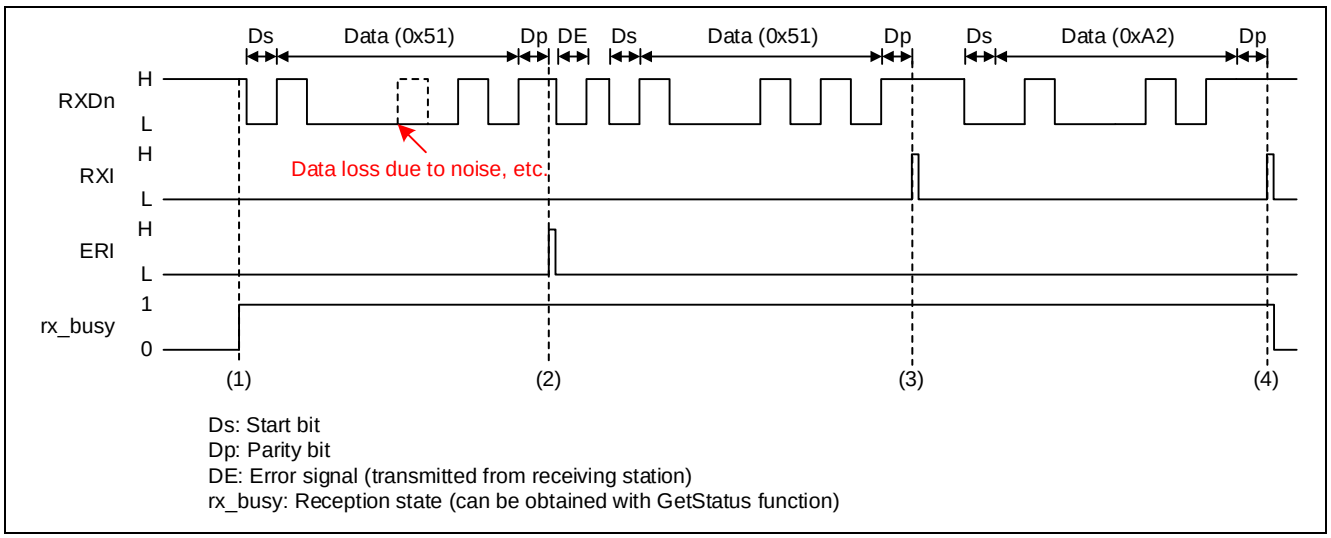


Figure 3-46 Operation When Parity Error is Detected (2 bytes received, even parity)

(1) When the Receive function is executed, the rx_busy flag is set to "1" (reception busy).

(2) If the parity error is detected during data reception, the ERI interrupt is generated and the error signal is output automatically by hardware. (The received data is discarded and the RXI interrupt request is not generated.) In addition, the callback function is executed using ARM_USART_EVENT_RX_PARITY_ERROR as an argument if the callback function has been registered.

(3) If the re-transmitted data is received without a parity error detected, the RXI interrupt is generated. The value is read from the receive data register (RDR) to the specified buffer in the interrupt handling process.

(4) At the RXI interrupt after the last data is read, the rx_busy flag is cleared to "0" (reception wait state). If a callback function has been registered, the function is executed using ARM_USART_EVENT_ RECEIVE_COMPLETE as an argument. (Note)

Note. If the error signal is received after the last data is output, the ERI interrupt is generated and the error signal is output as well like in step (2).

3.5 Configurations

For USART driver, configuration definitions that can be modified by the user are provided in the r_usart_cfg.h file.

3.5.1 Transmission Control

This sets what to use for controlling transmission.

Name: SCIn_TRANSMIT_CONTROL (n = 0 to 5, 9)

Table 3-7 Settings of SCIn_TRANSMIT_CONTROL

Setting	Description
SCI_USED_INTERRUPT (initial value)	Uses interrupts for transmission control
SCI_USED_DMAC0	Uses DMAC0 for transmission control
SCI_USED_DMAC1	Uses DMAC1 for transmission control
SCI_USED_DMAC2	Uses DMAC2 for transmission control
SCI_USED_DMAC3	Uses DMAC3 for transmission control
SCI_USED_DTC	Uses DTC for transmission control

3.5.2 Reception Control

This sets what to use for controlling reception.

Name: SCIn_RECEIVE_CONTROL (n = 0 to 5, 9)

Table 3-8 Settings of SCIn_RECEIVE_CONTROL

Setting	Description
SCI_USED_INTERRUPT (initial value)	Uses interrupts for reception control
SCI_USED_DMAC0	Uses DMAC0 for reception control
SCI_USED_DMAC1	Uses DMAC1 for reception control
SCI_USED_DMAC2	Uses DMAC2 for reception control
SCI_USED_DMAC3	Uses DMAC3 for reception control
SCI_USED_DTC	Uses DTC for reception control

3.5.3 TDRE Check Timeout Time

This sets timeout time for transmit buffer empty wait at the start of DMAC/DTC transmission (0 to 65535)

Name: SCI_CHECK_TDRE_TIMEOUT

Initial value: (10)

3.5.4 TXI Interrupt Priority Level

This sets the priority level of the TXIn interrupt. (n = 0 to 5, 9)

Name: SCIn_TXI_PRIORITY

Table 3-9 Settings of SCIn_TXI_PRIORITY

Setting	Description
0	Sets the interrupt priority level to 0. (highest priority)
1	Sets the interrupt priority level to 1.
2	Sets the interrupt priority level to 2.
3 (initial value)	Sets the interrupt priority level to 3.

3.5.5 RXI Interrupt Priority Level

This sets the priority level of the RXIn interrupt. (n = 0 to 5, 9)

Name: SCIn_RXI_PRIORITY

Table 3-10 Settings of SCIn_RXI_PRIORITY

Setting	Description
0	Sets the interrupt priority level to 0. (highest priority)
1	Sets the interrupt priority level to 1.
2	Sets the interrupt priority level to 2.
3 (initial value)	Sets the interrupt priority level to 3.

3.5.6 ERI Interrupt Priority Level

This sets the priority level of the ERIn interrupt. (n = 0 to 5, 9)

Name: SCIn_ERI_PRIORITY

Table 3-11 Settings of SCIn_ERI_PRIORITY

Setting	Description
0	Sets the interrupt priority level to 0. (highest priority)
1	Sets the interrupt priority level to 1.
2	Sets the interrupt priority level to 2.
3 (initial value)	Sets the interrupt priority level to 3.

3.5.7 Definition of Software-Controlled CTS Pin

These define the software-controlled CTS pin.

Name: USARTn_CTS_PORT (Note 1), USARTn_CTS_PIN (n = 0 to 5, 9)

Table 3-12 Settings of USARTn_CTS_PORT and USARTn_CTS_PIN

Name	Initial Value	Description
USARTn_CTS_PORT (Note)	(PORT0->PIDR)	Sets CTS pin as PORT0.
USARTn_CTS_PIN	0	Sets CTS pin as PORTi00. ("i" indicates a port number specified with USARTn_CTS_PORT.)

Note. This definition is commented out by default.
When using the software-controlled CTS pin, uncomment the definition.

3.5.8 Definition of RTS Pin under Software Control

This define the software-controlled RTS pin.

Name: USARTn_RTS_PORT (Note 2), USARTn_RTS_PIN (n = 0 to 5, 9)

Table 3-13 Settings of USARTn_RTS_PORT、USARTn_RTS_PIN

Name	Initial Value	Description
USARTn_RTS_PORT (Note 2)	(PORT0->PODR)	Sets RTS pin as PORT0.
USARTn_RTS_PIN	0	Sets RTS pin as PORTi00. ("i" indicates a port number specified with USARTn_RTS_PORT.)

Note. This definition is commented out by default.
When using the software-controlled RTS pin, uncomment the definition.

3.5.9 Function Allocation to RAM

This initializes the settings for executing specific functions of the USART driver via RAM.

This configuration definition for setting function allocation to RAM has function-specific definitions.

Name: USART_CFG_SECTION_xxx

A function name xxx should be written in all capital letters.

Example: ARM_USART_INITIALIZE function → USART_CFG_SECTION_ARM_USART_INITIALIZE

Table 3-14 Settings of USART_CFG_SECTION_xxx

Setting	Description
SYSTEM_SECTION_CODE	Does not allocate the function to RAM.
SYSTEM_SECTION_RAM_FUNC	Allocates the function to RAM.

Table 3-15 Initial State of Function Allocation to RAM

No.	Function Name	Allocation to RAM
1	ARM_USART_GetVersion	
2	ARM_USART_GetCapabilities	
3	ARM_USART_Initialize	
4	ARM_USART_Uninitialize	
5	ARM_USART_PowerControl	
6	ARM_USART_Send	
7	ARM_USART_Receive	
8	ARM_USART_Transfer	
9	ARM_USART_GetTxCount	
10	ARM_USART_GetRxCount	
11	ARM_USART_Control	
12	ARM_USART_GetStatus	
13	ARM_USART_SetModemControl	
14	ARM_USART_GetModemStatus	
15	scin_txi_interrupt (n = 0 to 5, 9) (TXI interrupt handling process)	✔
16	scin_rxi_interrupt (n = 0 to 5, 9) (RXI interrupt handling process)	✔
17	scin_eri_interrupt (n = 0 to 5, 9) (ERI interrupt handling process)	✔

4. Detailed Information of Driver

This chapter describes the detailed specifications implementing the functions of this driver.

4.1 Function Specifications

The specifications and processing flow of each function of the USART driver are described in this section.

In the flow of processing, some decision methods such as conditional branching are omitted, and consequently the flowcharts do not exactly show the actual processing.

4.1.1 ARM_USART_GetVersion Function

Table 4-1 ARM_USART_GetVersion Function Specifications

Format	ARM_DRIVER_VERSION ARM_USART_GetVersion(void)
Description	Acquires USART driver version.
Argument	None
Return value	USART driver version

Remarks	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	ARM_DRIVER_VERSION version;
	version = sci0Drv->GetVersion();
	
	}

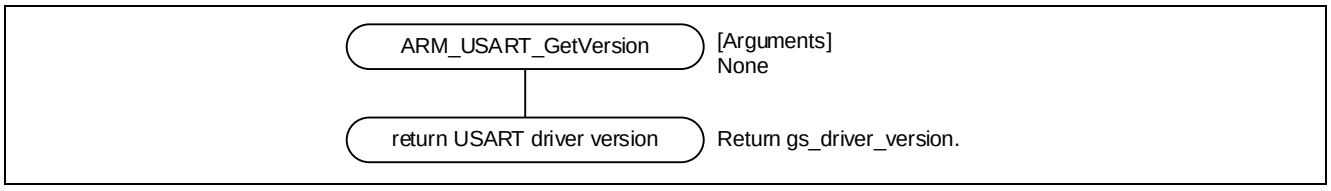


Figure 4-1 ARM_USART_GetVersion Function Processing Flow

4.1.2 ARM_USART_GetCapabilities Function

Table 4-2 ARM_USART_GetCapabilities Function Specifications

Format	ARM_USART_CAPABILITIES ARM_USART_GetCapabilities(void)
Description	Acquires USART driver functions.
Argument	Note
Return value	Driver functions

Remarks	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	ARM_USART_CAPABILITIES cap;
	cap = sci0Drv->GetCapabilities();
	
	}

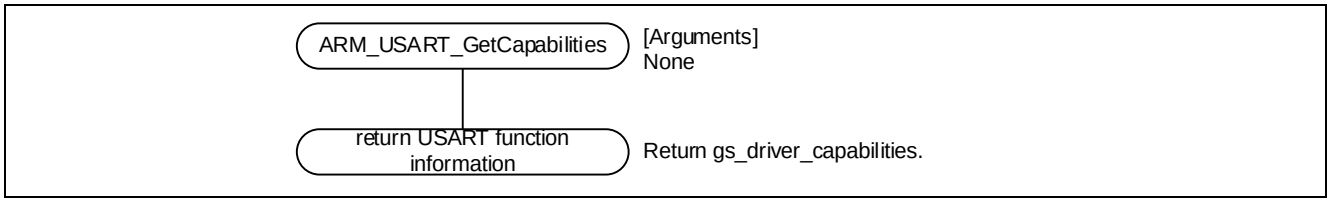


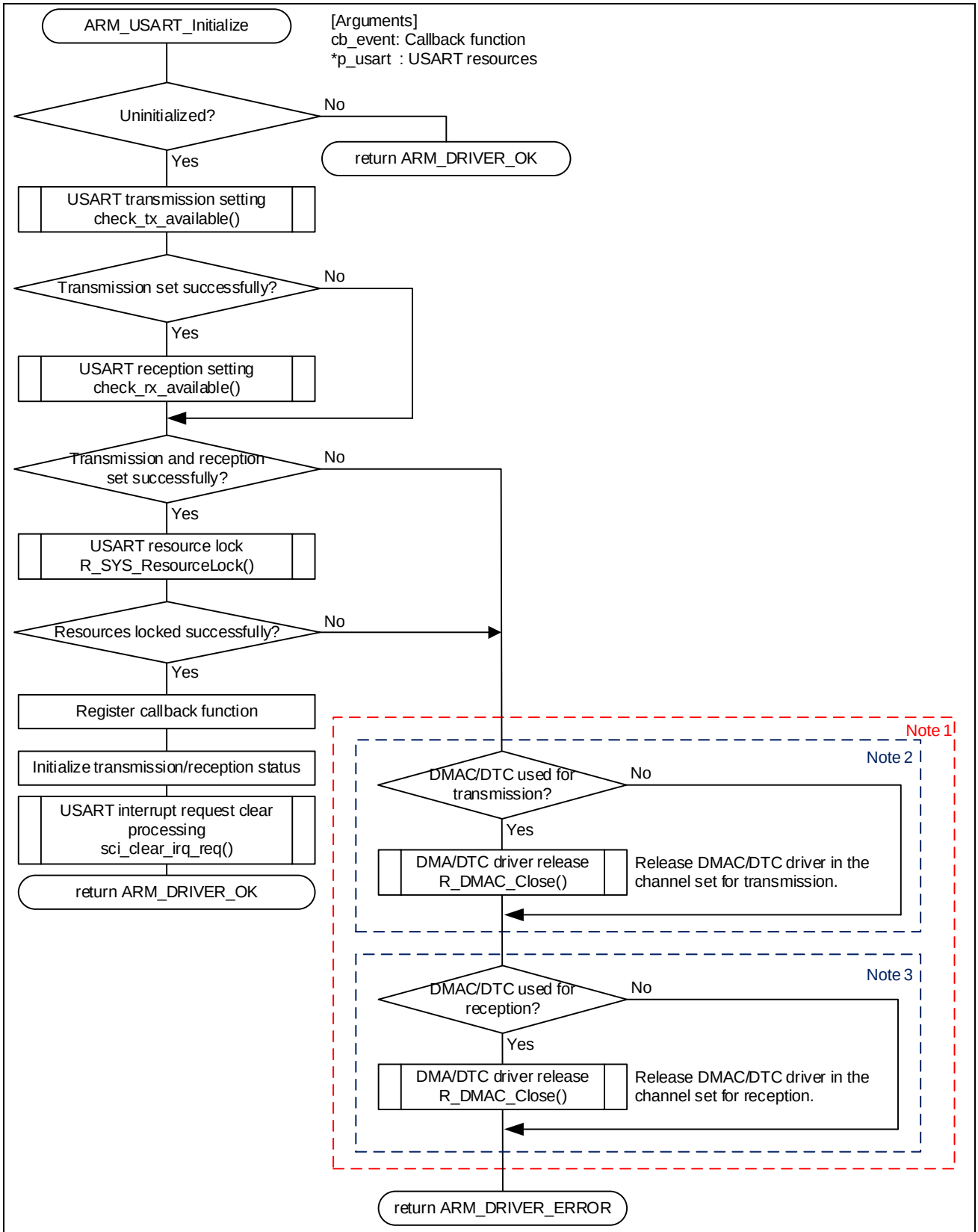
Figure 4-2 ARM_USART_GetCapabilities Function Processing Flow

4.1.3 ARM_USART_Initialize Function

Table 4-3 ARM_USART_Initialize Function Specifications

Format	int32_t ARM_USART_Initialize(ARM_USART_SignalEvent_t cb_event, st_usart_resources_t * const p_usart)
Description	Initializes USART driver (initializes RAM, makes register settings, and registers interrupts to NVIC).
Argument	ARM_USART_SignalEvent_t cb_event: Callback function Specifies the callback function to be executed when an event occurs. If NULL is set, the callback function will not be executed.
	st_usart_resources_t * const p_usart : Resources of USART Specify the resources of the USART to initialize.
Return value	ARM_DRIVER_OK USART initialization completed
	ARM_DRIVER_ERROR USART initialization failed If one of the following conditions is detected, initialization will fail. • If neither transmission nor reception can be used (due to an errorneous setting in communication control, NVIC registration, etc.) • If the resources of an SCI channel to use is locked (If SCIn is already locked by the R_SYS_ResourceLock function

Remarks	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	static void callback(uint32_t event);
	
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	sci0Drv->Initialize(callback);
	}



Note 1. When DMAC or DTC function is set to use for a channel with r_usart_cfg.h, the appropriate processing is enabled.
Note 2. When DMAC or DTC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled.
Note 3. When DMAC or DTC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled.

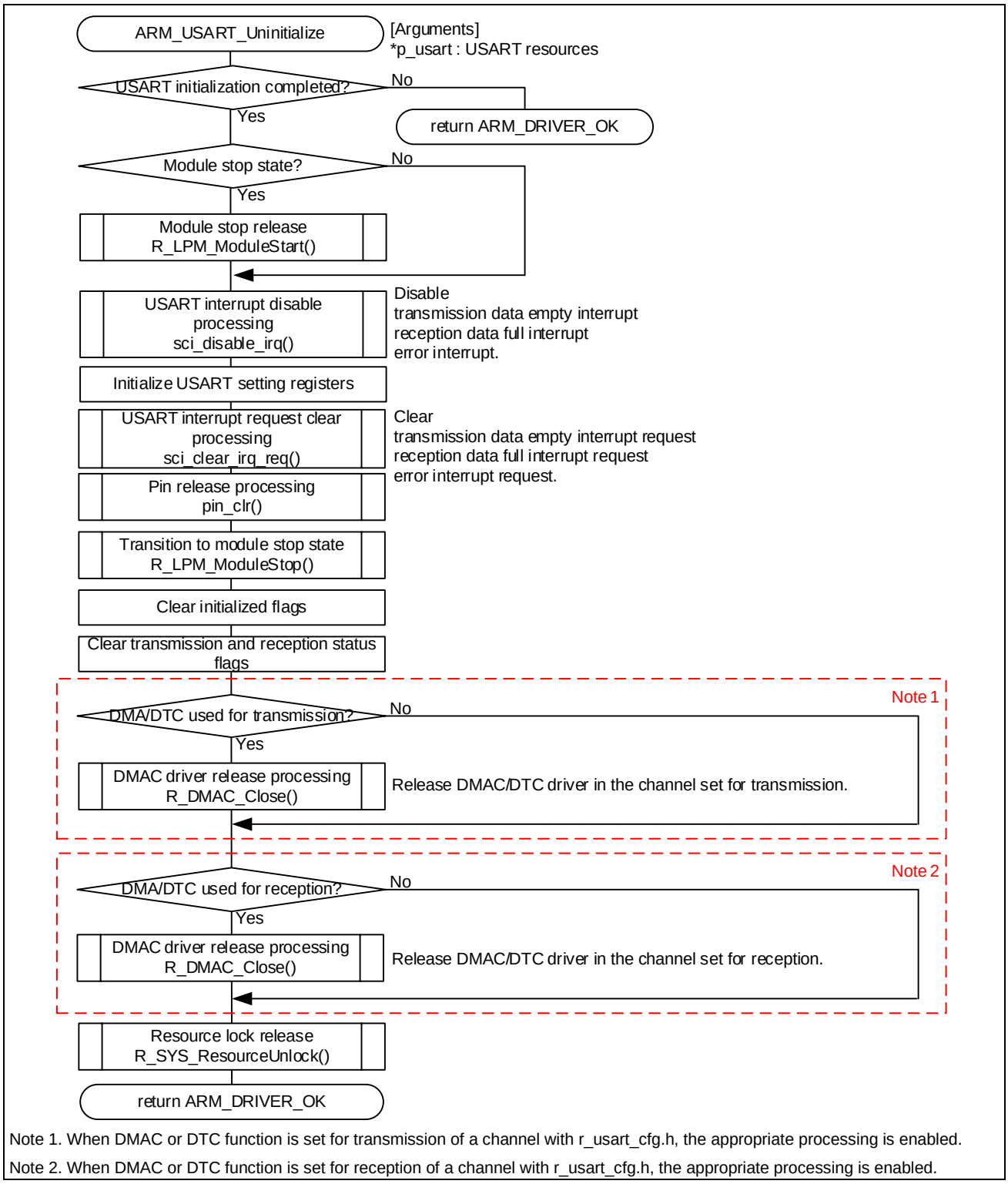
Figure 4-3 ARM_USART_Initialize Function Processing Flow

4.1.4 ARM_USART_Uninitialize Function

Table 4-4 ARM_USART_Uninitialize Function Specifications

Format	int32_t ARM_USART_Uninitialize(st_usart_resources_t * const p_usart)
Description	Releases the USART driver.
Argument	st_usart_resources_t * const p_usart: Resources of USART Specifies the resources of the USART to be released.
Return value	ARM_DRIVER_OK USART released normally

Remarks	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	sci0Drv->Uninitialize();
	}



Note 1. When DMAC or DTC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled.

Note 2. When DMAC or DTC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled.

Figure 4-4 ARM_USART_Uninitalize Function Processing Flow

4.1.5 ARM_USART_PowerControl Function

Table 4-5 ARM_USART_PowerControl Function Specifications

Format	int32_t ARM_USART_PowerControl(ARM_POWER_STATE state, st_usart_resources_t * const p_usart)
Description	Releases the USART from the module stop state or causes a transition to the mode.
Argument	ARM_POWER_STATE state: Power setting Set one of the following. ARM_POWER_OFF: Causes a transition to the module stop state. ARM_POWER_FULL: Releases the USART from the module stop state. ARM_POWER_LOW: This setting is not supported.
	st_usart_resources_t * const p_usart: Resources of USART Specifies the resources of the USART to which power is supplied.
Return value	ARM_DRIVER_OK Power setting change completed
	ARM_DRIVER_ERROR Power setting change failed If one of the following conditions is detected, the power setting change will fail. • If this function is executed with USART uninitialized • If transition to the module stop state has failed (An error has occurred in R_LPM_ModuleStart)
	ARM_DRIVER_ERROR_UNSUPPORTED Specified power setting is not supported

Remarks	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	sci0Drv->PowerControl(ARM_POWER_FULL);
	}

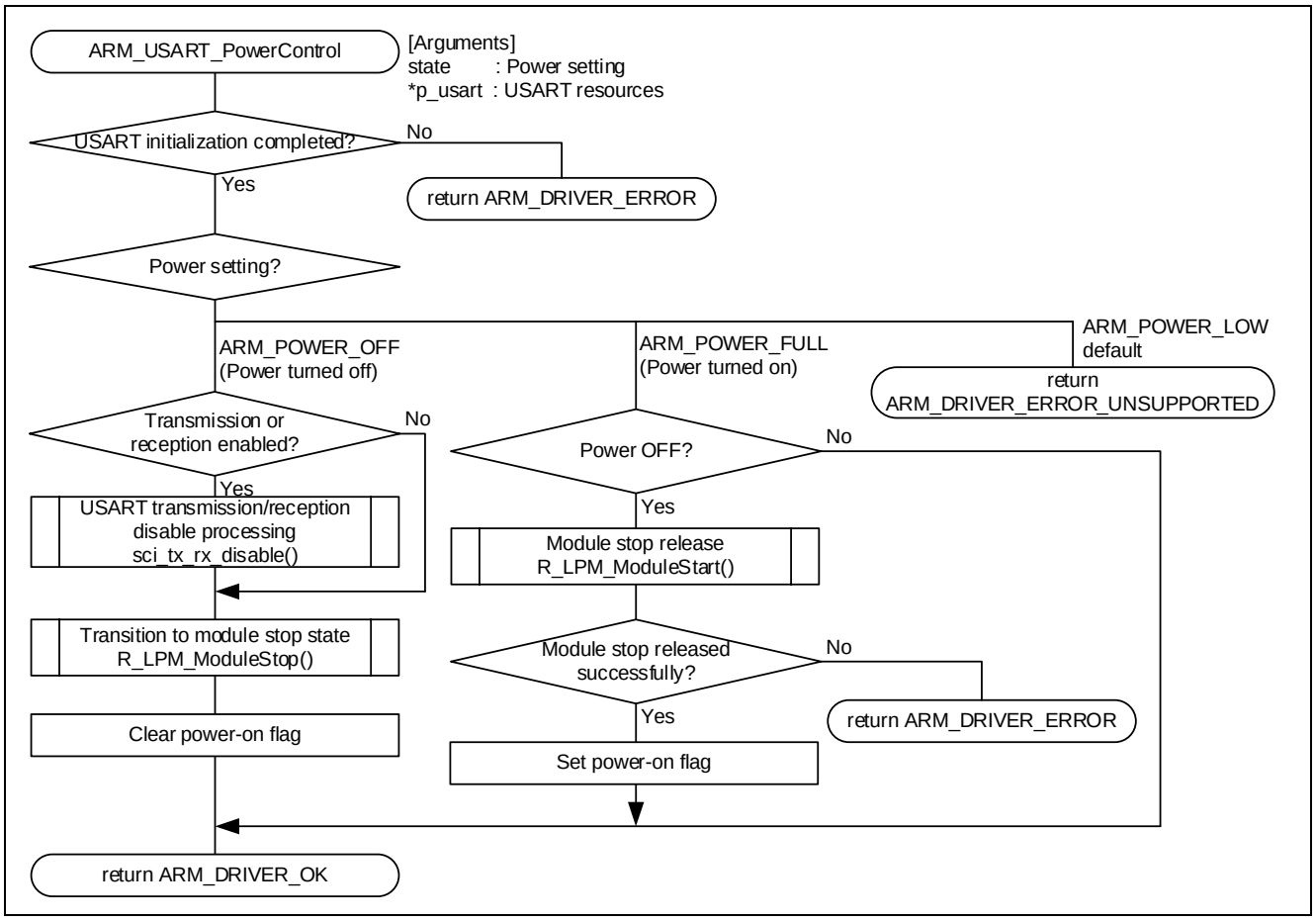


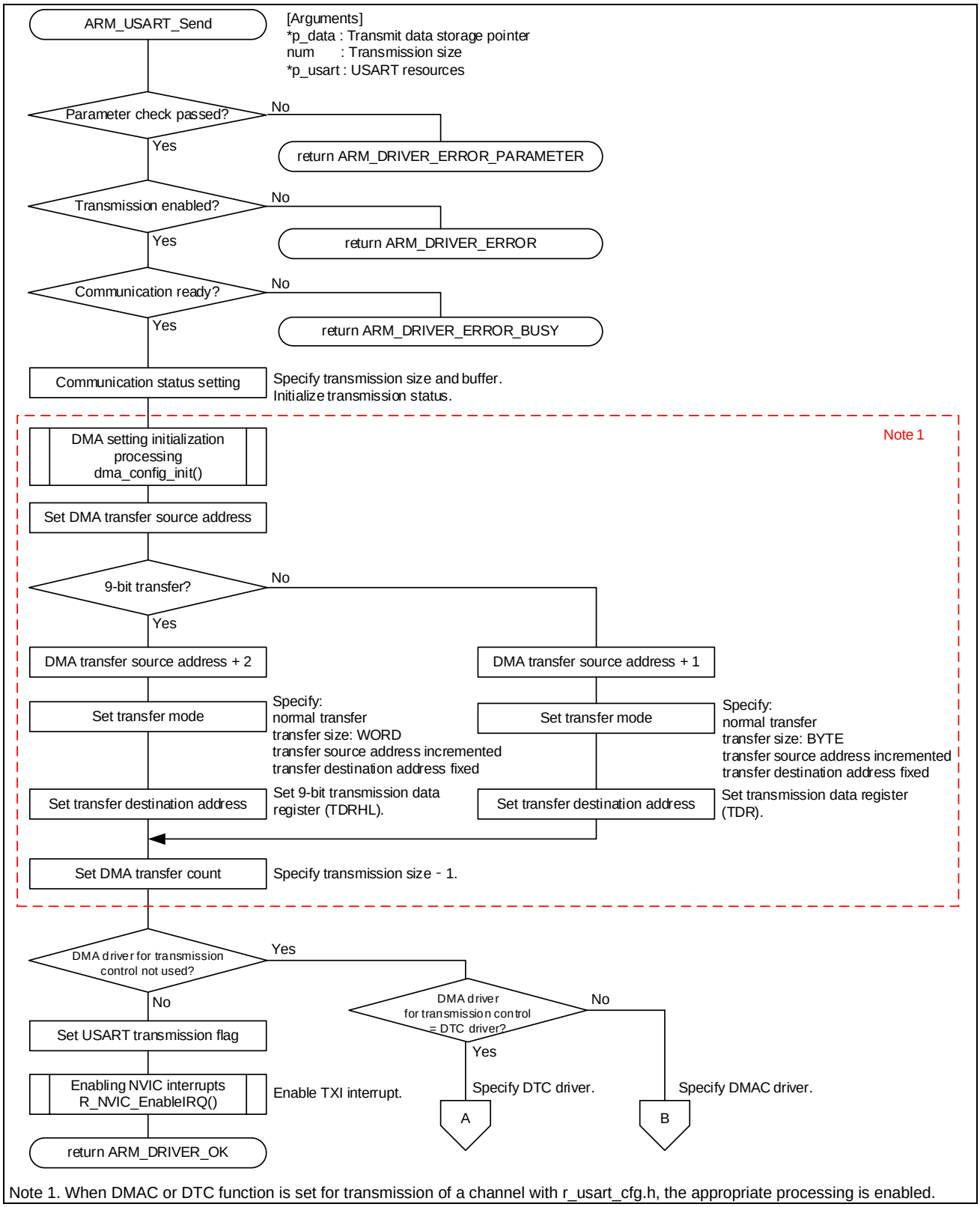
Figure 4-5 ARM_USART_PowerControl Function Processing Flow

4.1.6 ARM_USART_Send Function

Table 4-6 ARM_USART_Send Function Specifications

Format	int32_t ARM_USART_Send(void const * const p_data, uint32_t num, st_usart_resources_t * const p_usart)
Description	Starts transmission.
Argument	void const * const *p_data: Pointer to transmit data storage Specifies the start address of the buffer where data to be transmitted is stored.
	uint32_t num: Transmission size Specifies the size of data to be transmitted.
	st_usart_resources_t * const p_usart : Resources of USART Specifies the resources of the USART that transmit data.
Return value	ARM_DRIVER_OK Transmission started normally
	ARM_DRIVER_ERROR Transmission start failed If one of the following conditions is detected, the transmission start will fail • If this function is executed with transmission disabled • If DMAC/DTC is specified for transmission and DMA driver setting has failed
	ARM_DRIVER_ERROR_BUSY Transmission failed because of busy state If one of the following conditions is detected, the transmission will fail because of a busy state. • If judged as transmission in progress • If judged as reception in progress in clock synchronous mode • If DMAC/DTC is used for transmission and transmit buffer empty wait has timed out
	ARM_DRIVER_ERROR_PARAMETER Parameter error If one of the following settings is specified, a parameter error will occur. • The pointer to transmit data storage is set to NULL. • The transmission size is set to 0

Remarks	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	const uint8_t tx_data[2] = {0x51, 0xA2};
	
	main()
	{
	sci0Drv->Send(&tx_data[0], 2);
	}



Note 1. When DMAC or DTC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled.

Figure 4-6 ARM_USART_Send Function Processing Flow (1/3)

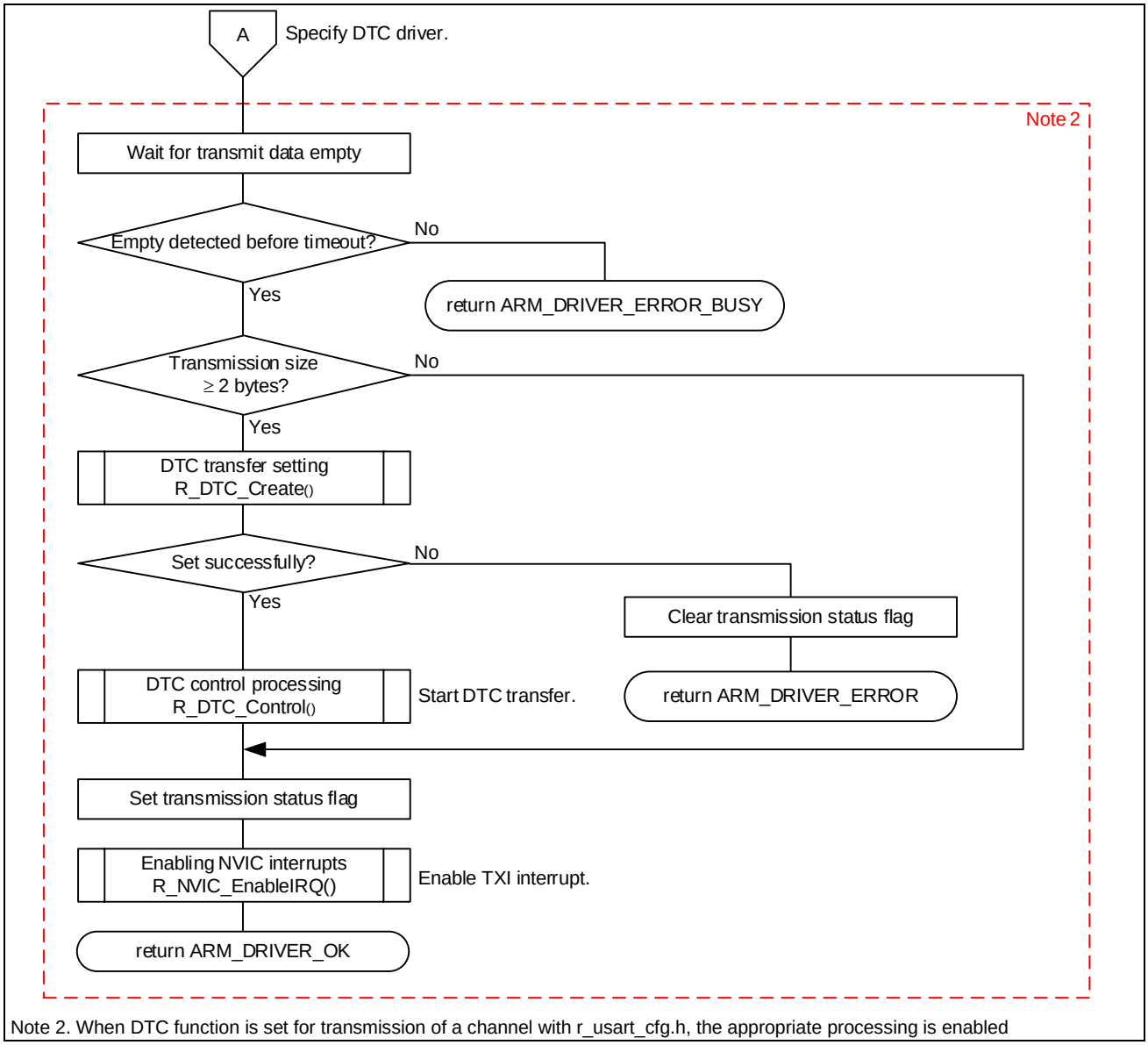
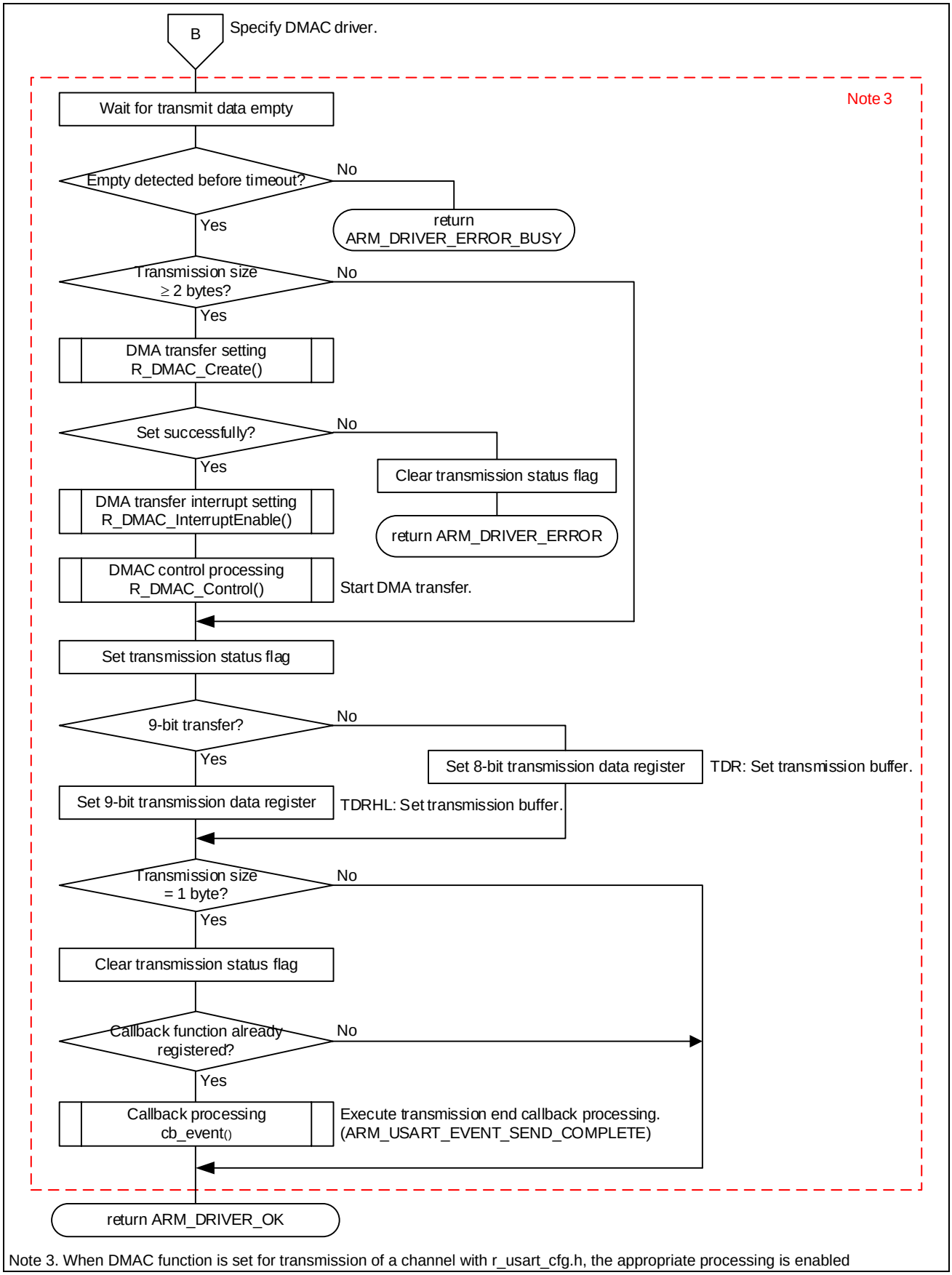


Figure 4-7 ARM_USART_Send Function Processing Flow (2/3)



Note 3. When DMAC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled

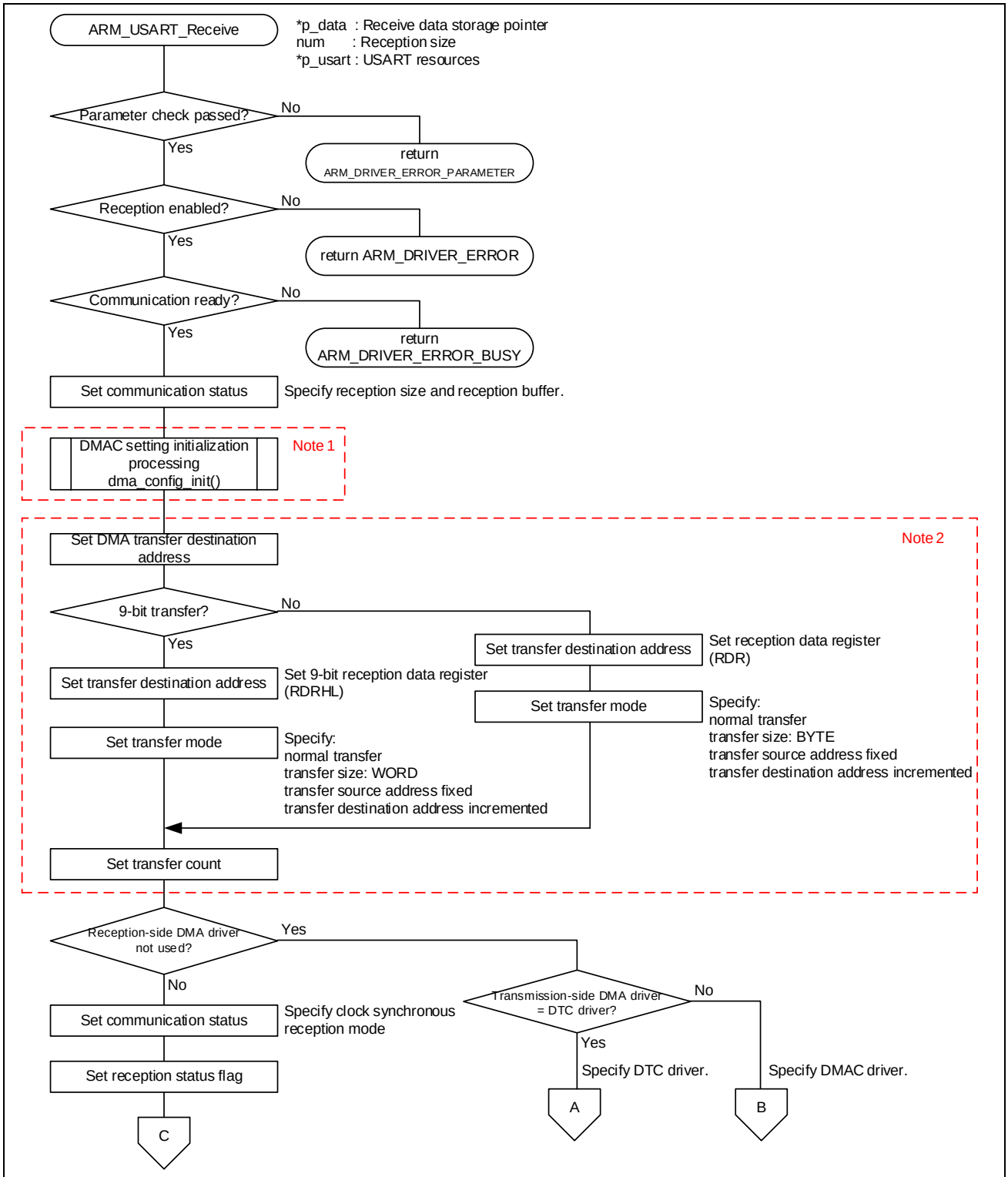
Figure 4-8 ARM_USART_Send Function Processing Flow (3/3)

4.1.7 ARM_USART_Receive Function

Table 4-7 ARM_USART_Receive Function Specifications

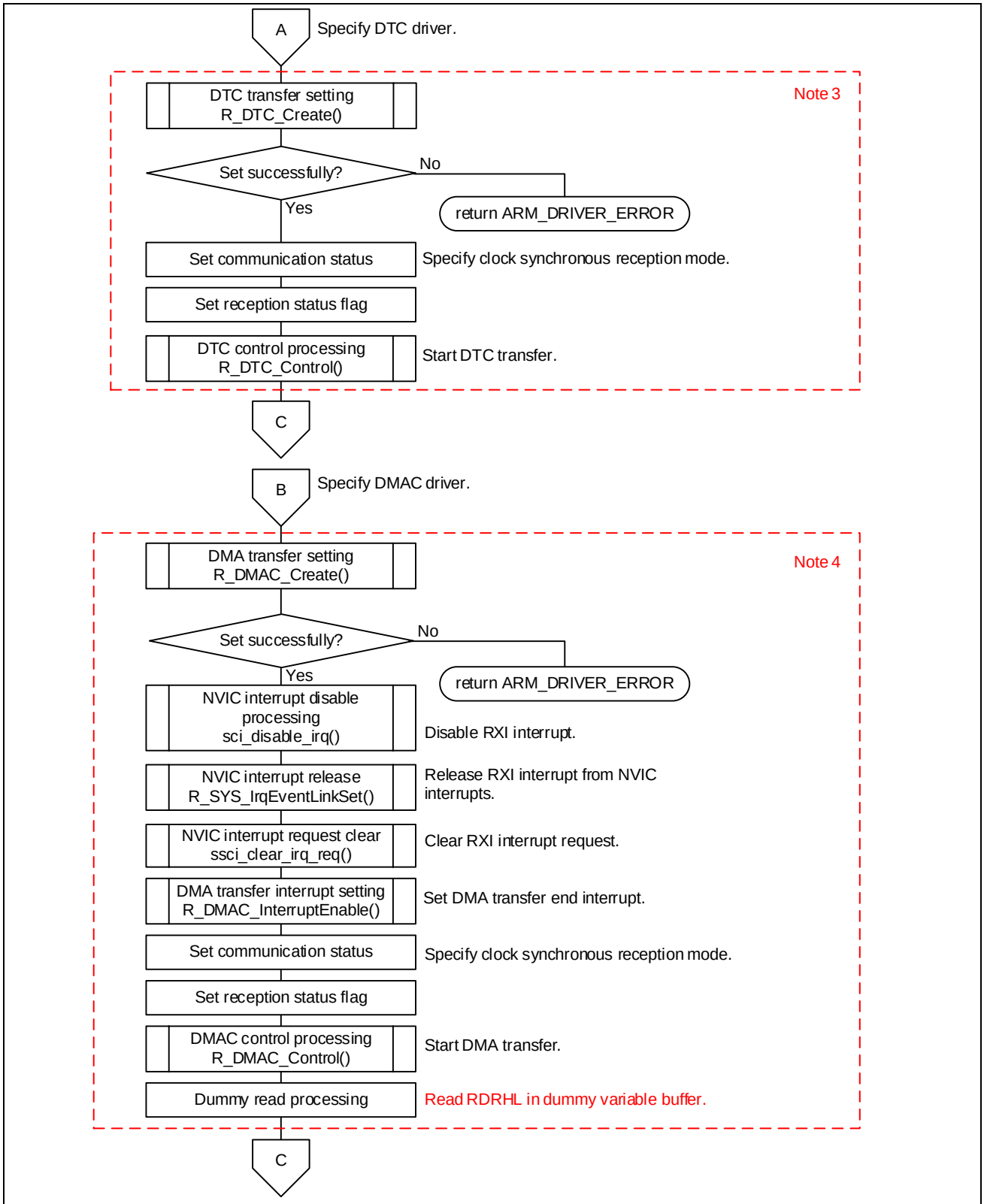
Format	int32_t ARM_USART_Receive(void * const p_data, uint32_t num, st_usart_resources_t * const p_usart)
Description	Starts reception.
Argument	void * const p_data: Pointer to receive data storage Specifies the start address of the buffer where received data is to be stored.
	uint32_t num : Reception size Specifies the size of data to be received.
	st_usart_resources_t * const p_usart : Resources of USART Specifies the resources of the USART that receive data.
Return value	ARM_DRIVER_OK Reception started normally
	ARM_DRIVER_ERROR Reception start failed If one of the following conditions is detected, the reception start will fail. • If this function is executed with reception disabled • If DMAC/DTC is specified for reception and DMA driver setting has failed
	ARM_DRIVER_ERROR_BUSY Reception failed because of busy state If one of the following conditions is detected, the reception will fail because of a busy state. • Judged as reception in progress • Judged as transmission in progress in clock synchronous mode • If DMAC/DTC is used for transmission and transmit buffer empty wait has timed out
	ARM_DRIVER_ERROR_PARAMETER Parameter error If one of the following settings is specified, a parameter error will occur. • The pointer to receive data storage is set as NULL. • The receive data size is set to 0.

Remarks	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	uint8_t rx_data[2];
	
	main()
	{
	sci0Drv->Receive(&rx_data[0], 2);
	}



Note 1. When DMAC or DTC function is set to use for a channel with r_usart_cfg.h, the appropriate processing is enabled.
Note 2. When DMAC or DTC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled.

Figure 4-9 ARM_USART_Receive Function Processing Flow (1/5)



Note 3. When DTC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled
Note 4. When DMAC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled

Figure 4-10 ARM_USART_Receive Function Processing Flow (2/5)

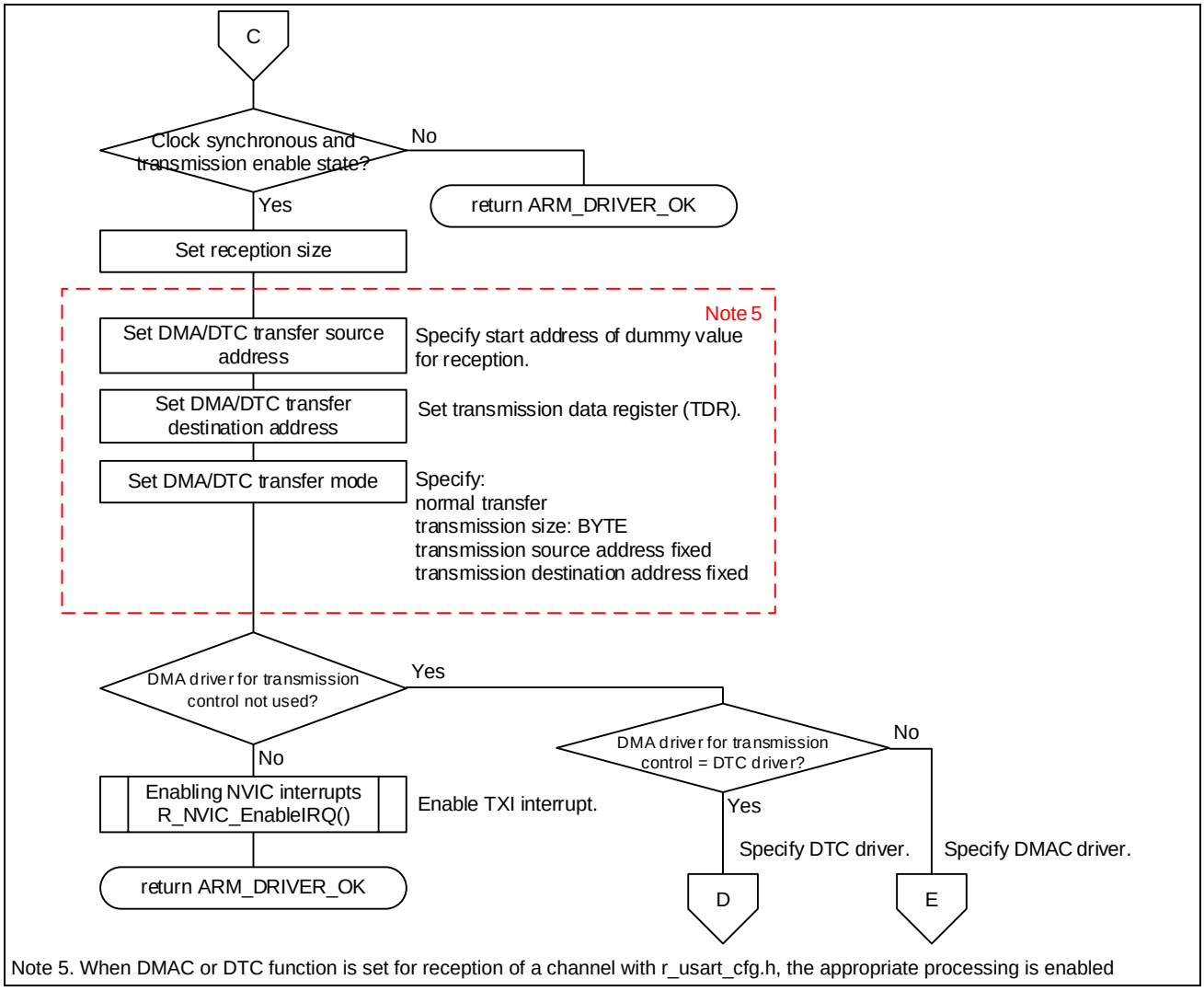


Figure 4-11 ARM_USART_Receive Function Processing Flow (3/5)

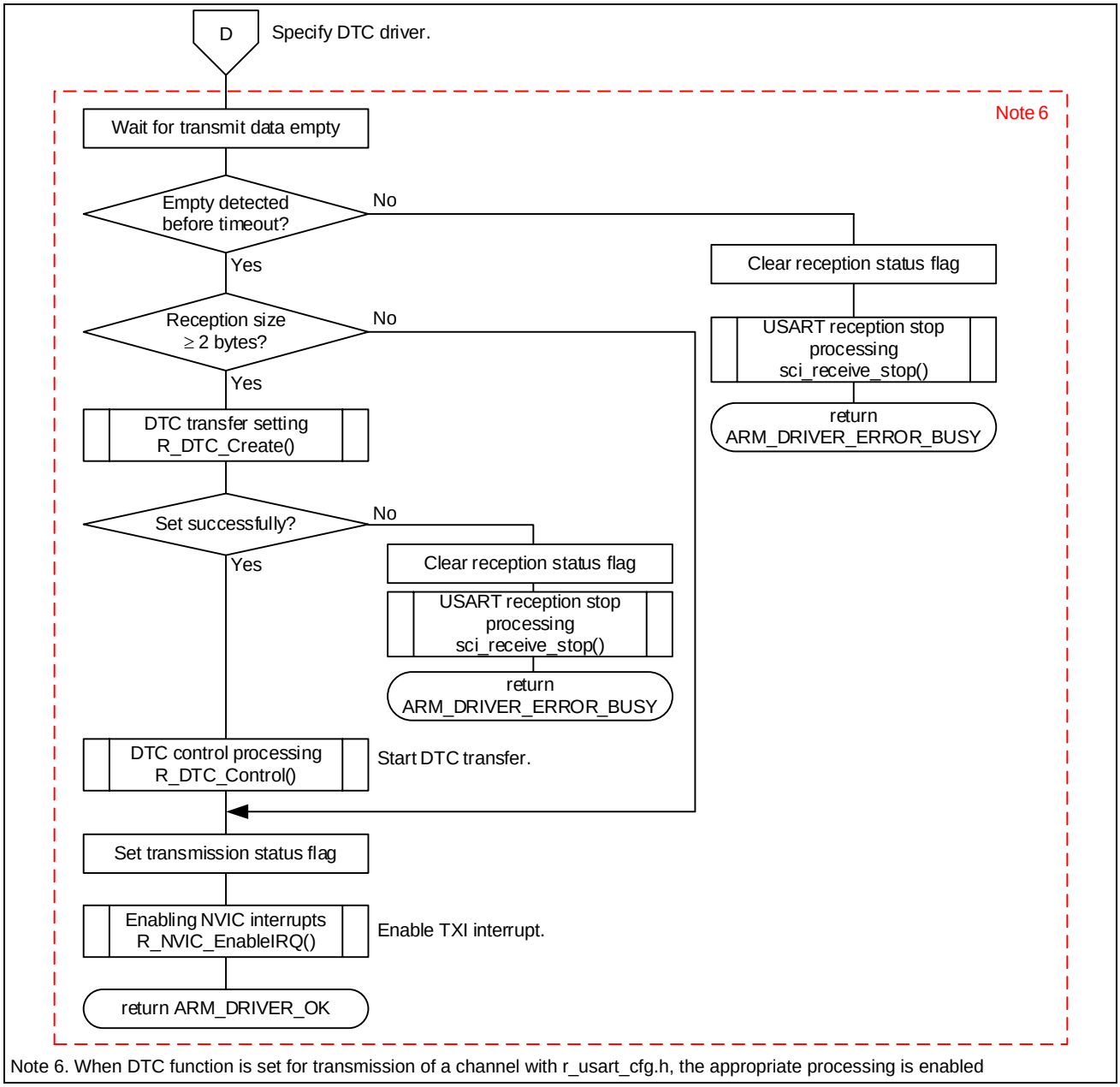


Figure 4-12 ARM_USART_Receive Function Processing Flow (4/5)

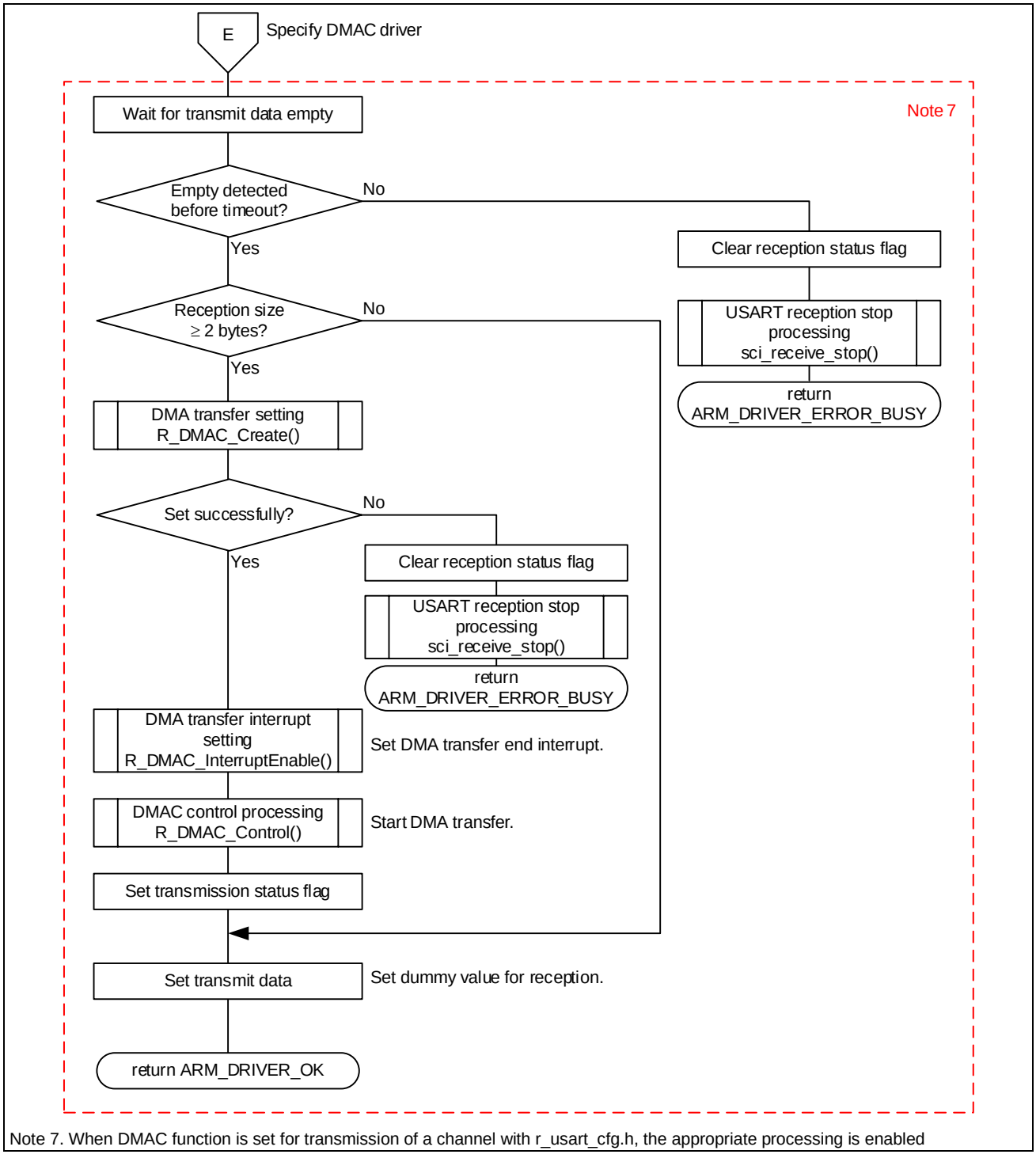


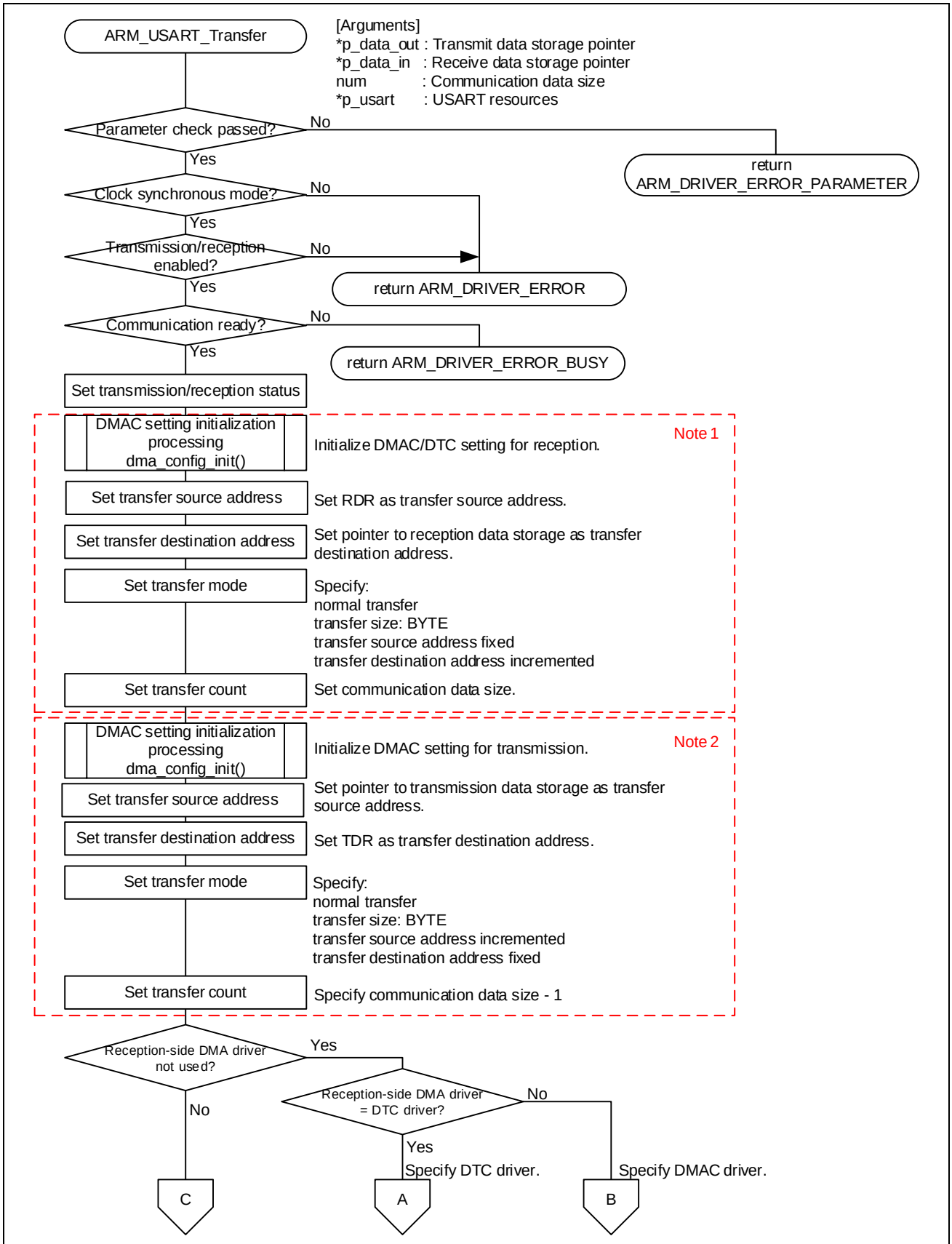
Figure 4-13 ARM_USART_Receive Function Processing Flow (5/5)

4.1.8 ARM_USART_Transfer Function

Table 4-8 ARM_USART_Transfer Function Specifications

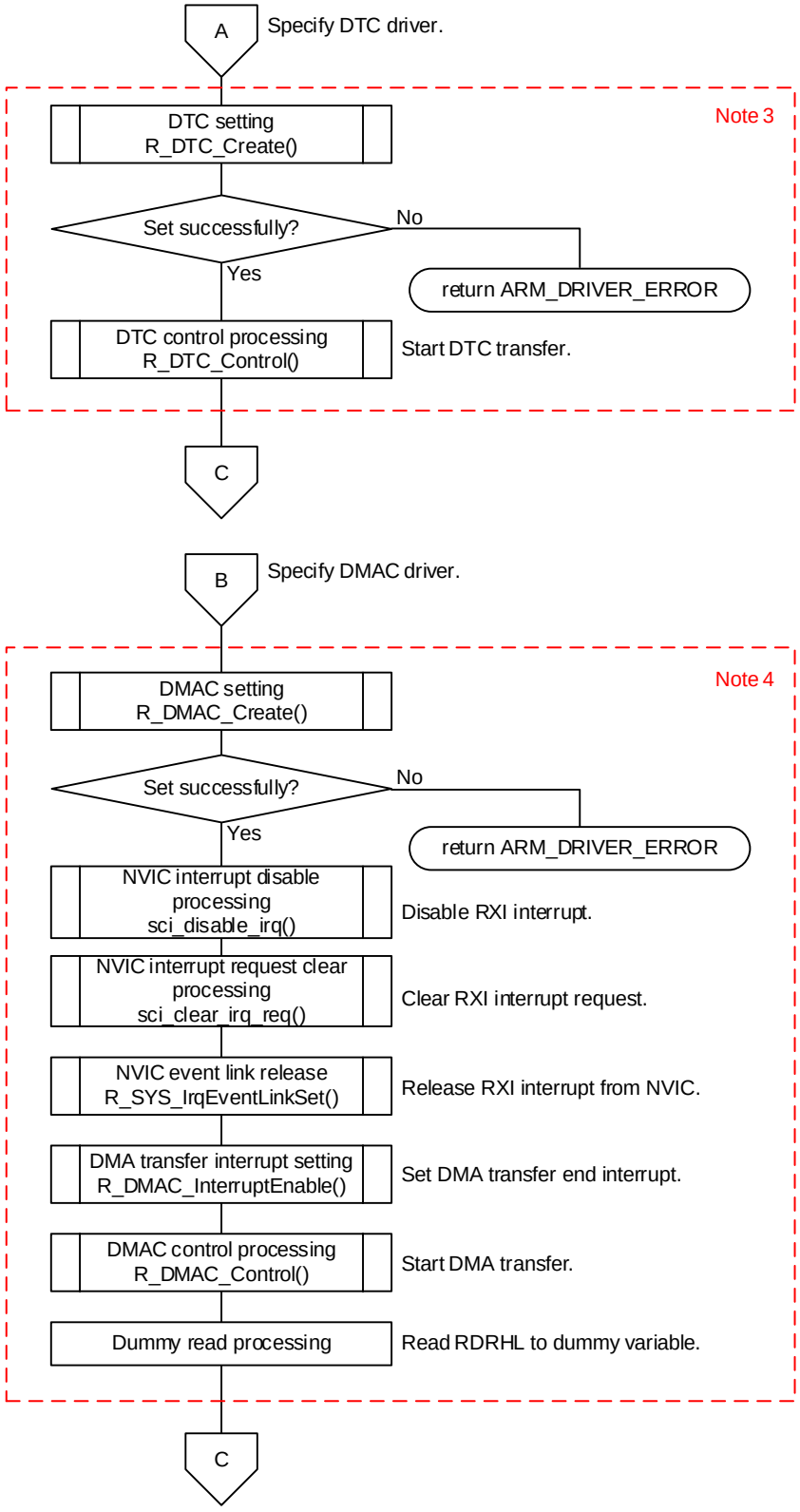
Format	int32_t ARM_USART_Transfer(void const * const p_data_out, void * const p_data_in, uint32_t num, st_usart_resources_t * const p_usart)
Description	Starts transmission and reception in clock synchronous mode.
Argument	void const * const p_data_out : Pointer to transmit data storage Specifies the start address of the buffer where data to be transmitted is stored.
	void * const p_data_in : Pointer to receive data storage Specifies the start address of the buffer where received data is stored.
	uint32_t num : Transmission/reception size Specifies the size of data to be transmitted and received.
	st_usart_resources_t * const p_usart : Resources of USART Specifies the resources of the USART that receive data.
Return value	ARM_DRIVER_OK Transmission/reception started normally
	ARM_DRIVER_ERROR Transmission/reception start failed If one of the following conditions is detected, the transmission/reception start will fail. • If not in clock synchronous mode • If this function is executed with transmission disabled or reception disabled • If DMAC/DTC is specified for transmission or reception and DMA driver setting has failed
	ARM_DRIVER_ERROR_BUSY Transmission failed because of busy state If one of the following conditions is detected, the reception will fail because of a busy state. • If judged as reception in progress or transmission in progress • If DMAC/DTC is used for transmission and transmit buffer empty wait has timed out
	ARM_DRIVER_ERROR_PARAMETER Parameter error If one of the following conditions is detected, a parameter error will occur. • The transmission/reception size is set to 0 • The pointer to transmit data storage is set as NULL. • The pointer to receive data storage is set as NULL

Remarks	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	uint8_t rx_data[2];
	const uint8_t tx_data[2] = {0x51, 0xA2};
	
	main()
	{
	sci0Drv->Transfer (&tx_data[0], &rx_data[0], 2);
	}



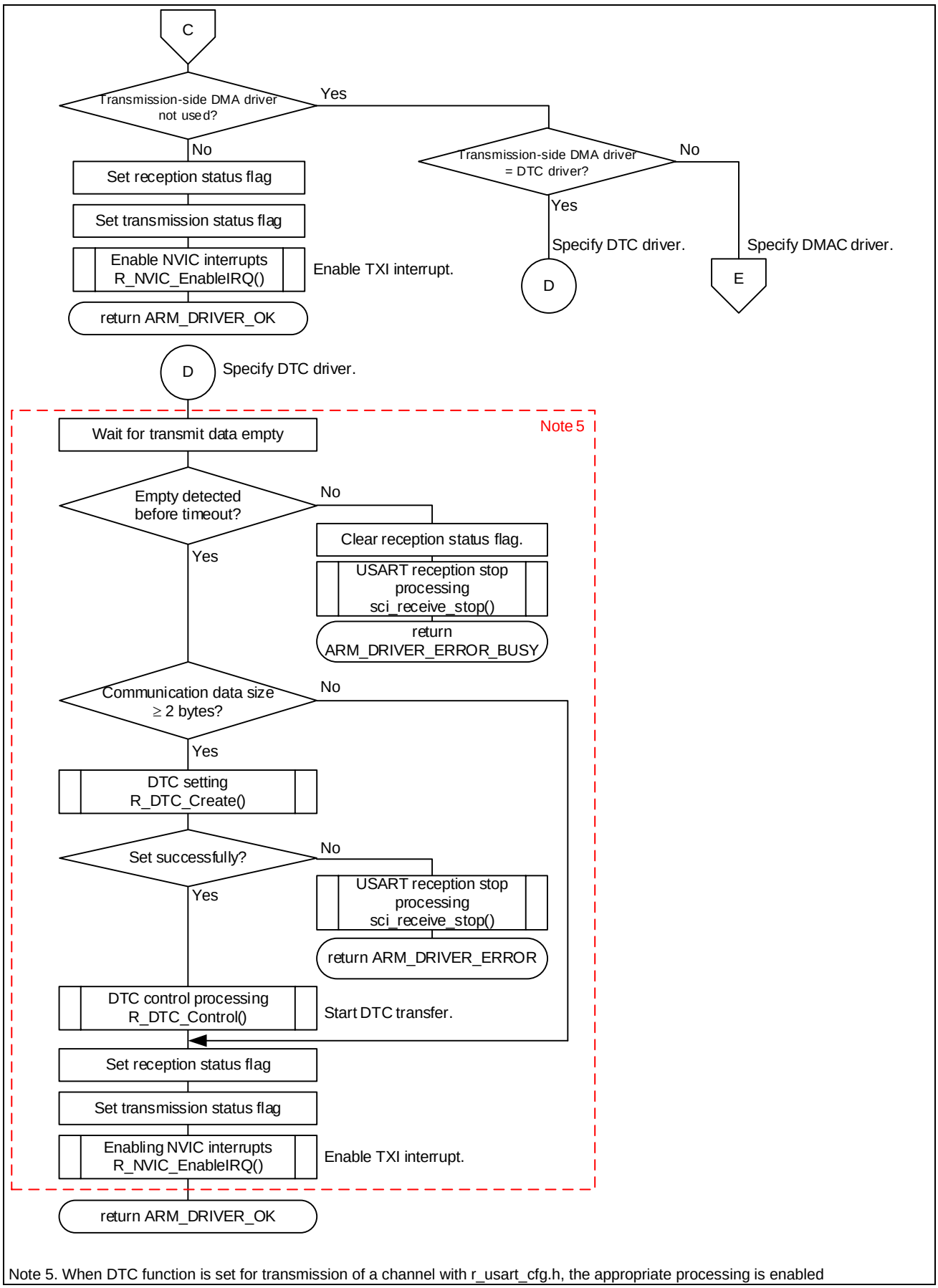
Note 1. When DMAC or DTC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled
Note 2. When DMAC or DTC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled

Figure 4-14 ARM_USART_Transfer Function Processing Flow (1/4)



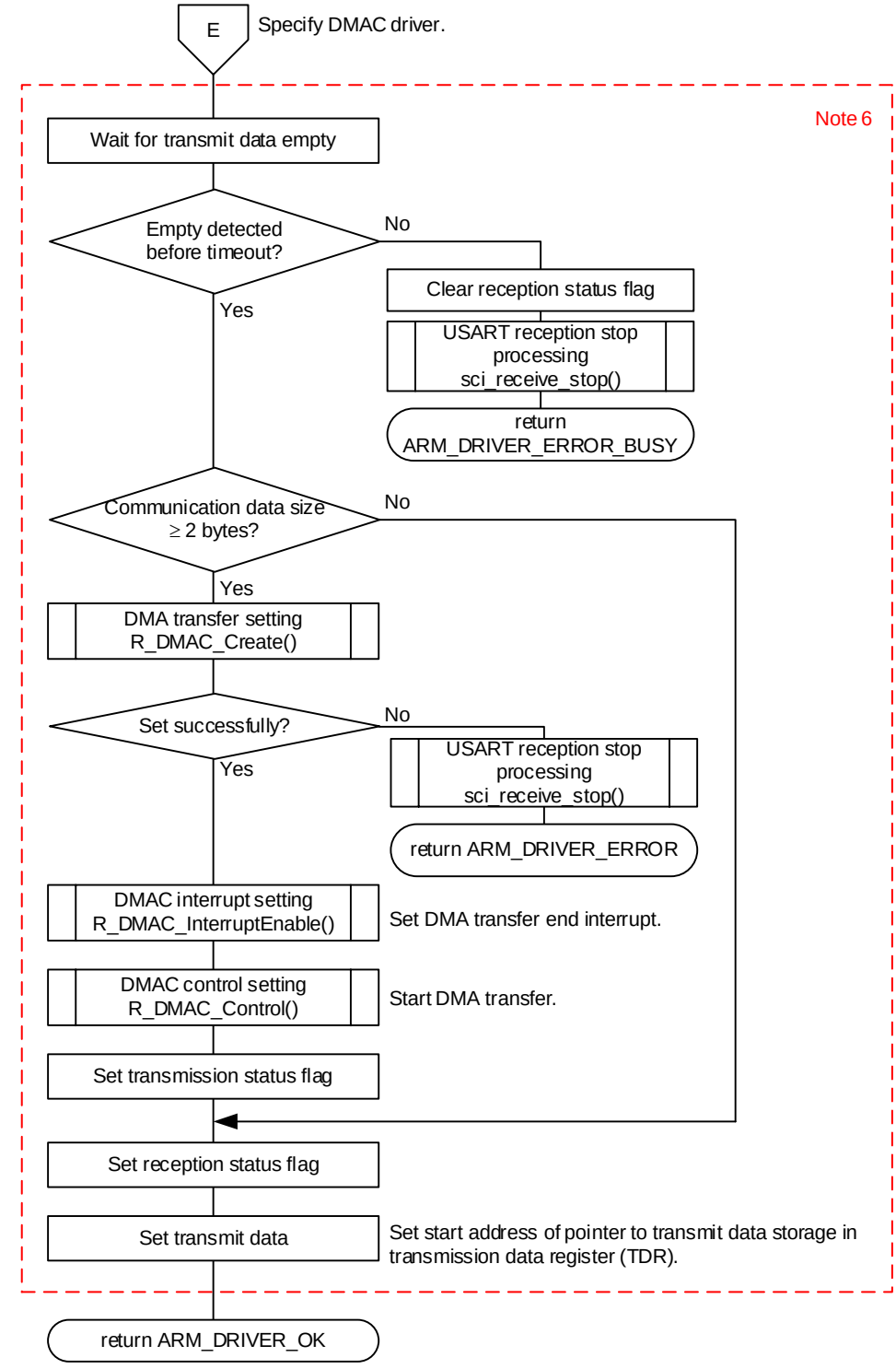
Note 3. When DTC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled
Note 4. When DMAC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled

Figure 4-15 ARM_USART_Transfer Function Processing Flow (2/4)



Note 5. When DTC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled

Figure 4-16 ARM_USART_Transfer Function Processing Flow (3/4)



Note 6. When DMAC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled

Figure 4-17 ARM_USART_Transfer Function Processing Flow (4/4)

4.1.9 ARM_USART_GetTxCount Function

Table 4-9 ARM_USART_GetTxCount Function Specifications

Format	uint32_t ARM_USART_GetTxCount(st_usart_resources_t const * const p_usart)
Description	Acquires the current transmission count.
Argument	st_usart_resources_t * const p_usart : Resources of USART Specifies the resources of the USART that get the transmission count.
Return value	Transmission count

Remarks	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	uint32_t tx_count;
	tx_count = sci0Drv->GetTxCount();
	}

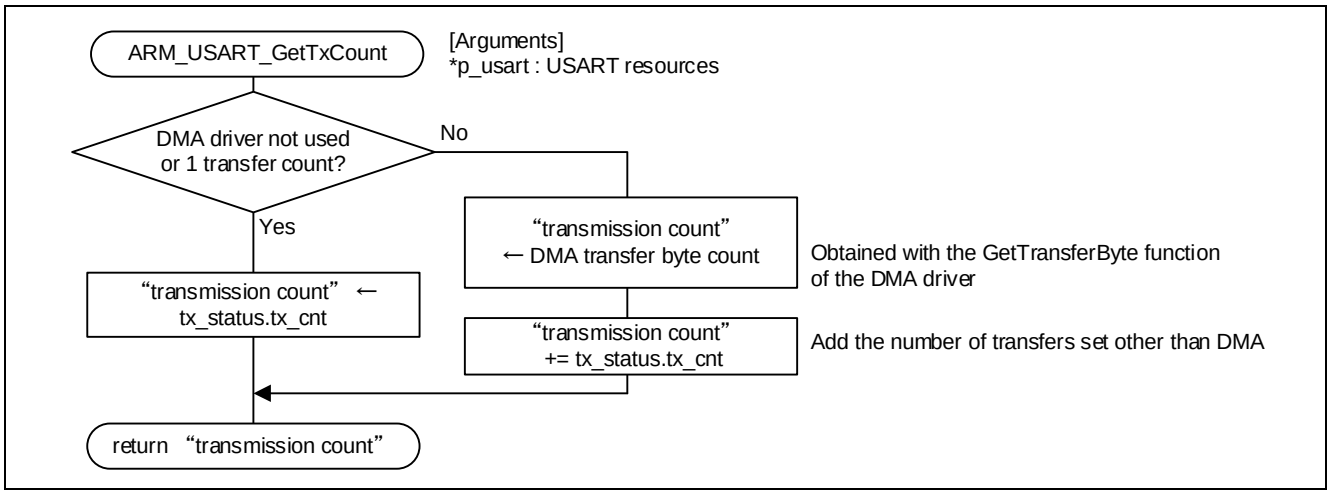


Figure 4-18 ARM_USART_GetTxCount Function Processing Flow

4.1.10 ARM_USART_GetRxCount Function

Table 4-10 ARM_USART_GetRxCount Function Specifications

Format	uint32_t ARM_USART_GetRxCount(st_usart_resources_t const * const p_usart)
Description	Acquires the current reception count.
Argument	st_usart_resources_t * const p_usart : Resources of USART Specifies the resources of the USART that get the reception count.
Return value	Reception count

Remarks	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	uint32_t rx_count;
	rx_count = sci0Drv->GetRxCount();
	}

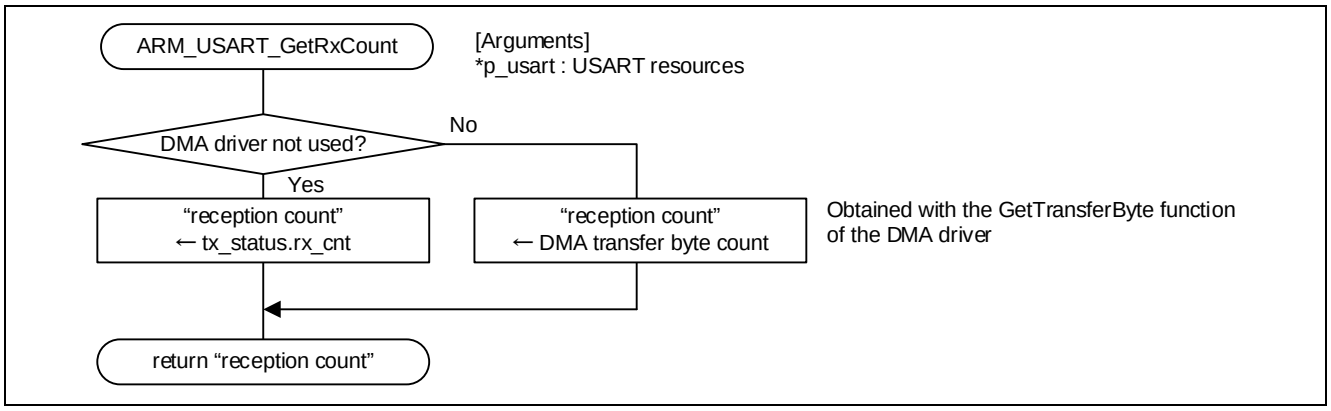


Figure 4-19 ARM_USART_GetRxCount Function Processing Flow

4.1.11 ARM_USART_Control Function

Table 4-11 ARM_USART_Control Function Specifications (1/2)

Format	int32_t ARM_USART_Control(uint32_t control, uint32_t arg, st_usart_resources_t const * const p_usart)
Description	Executes a control command of the USART.
Argument	uint32_t control : Control command See section 2.5.1 USART Control Command Control Command Definitions for the control commands.
	uint32_t arg : Command-specific argument (See Table 4-13 for the relationship between control commands and arguments.)
	st_usart_resources_t * const p_usart : Resources of USART Specifies USART resources to be controlled.
Return value	ARM_DRIVER_OK Control command execution completed
	ARM_DRIVER_ERROR Control command execution failed If one of the following conditions is detected, the control command execution will fail. • If this command is executed with the power supply turned off • If the second argument is set to 0 with asynchronous mode set (ARM_USART_MODE_ASYNCHRONOUS) • If the second argument is set to 0 with clock synchronous master mode set (ARM_USART_MODE_SYNCHRONOUS_MASTER) • If smart card mode (ARM_USART_MODE_SMART_CARD) is set without registering RXI and ERI interrupts to NVIC (ARM_USART_MODE_SMART_CARD) • If the same or a different operation mode attempts to be set with an operating mode being set • If a different baud rate is specified in smart card clock output setting (ARM_USART_SET_SMART_CARD_CLOCK) • If smart card clock output is executed in a mode other than smart card mode • If smart card clock output is executed with transmission and reception enabled. • If smart card NACK output setting (ARM_USART_CONTROL_SMART_CARD_NACK) is executed in a state other than enable state • If smart card NACK output setting is executed in a mode other than smart card mode • If transmission enable setting (ARM_USART_CONTROL_TX) is executed with transmission disabled • If reception enable setting (ARM_USART_CONTROL_RX) is executed with reception disabled. • If transmission and reception enable setting (ARM_USART_CONTROL_TX_RX) is executed with either transmission disabled or reception disabled • If reception enable setting is executed in clock synchronous mode and with transmission enabled • If transmission enable setting is executed in clock synchronous mode and with reception enabled • If transmission and reception enable setting is executed with only one of the transmission and reception enabled • If transmission enable setting, reception enable setting, or transmission and reception enable setting is executed before operation mode is set • If transmission suspend setting (ARM_USART_ABORT_SEND) is executed with transmission disabled • If reception suspend setting (ARM_USART_ABORT_RECEIVE) is executed with reception disabled • If transmission and reception suspend setting (ARM_USART_ABORT_TRANSFER) is executed with transmission disabled or reception disabled • If an illegal command is executed

Table 4-12 ARM_USART_Control Function Specifications (2/2)

Return value	ARM_USART_ERROR_DATA_BITS Data bit length setting error If the data bit length set for asynchronous mode is not 7, 8, or 9 bits, control command execution fails due to a data bit length setting error.
	ARM_USART_ERROR_PARITY Parity setting error
	If one of the following conditions is detected, a parity setting error will occur. • If the parity set for asynchronous mode is not any of no parity, odd parity, and even parity • If the parity set for smart card mode is not any of no parity, odd parity, and even parity
	ARM_USART_ERROR_STOP_BITS Stop bit length setting error If the stop bit length set for asynchronous mode is not 1 or 2 bits, a stop bit length setting error occurs.
	ARM_USART_ERROR_FLOW_CONTROL Flow control setting error If one of the following conditions is detected, a flow control setting error will occur. • If the flow control set for asynchronous mode is not any of no flow control, CTS control, and RTS control • If the flow control set for clock synchronous mode is not any of no flow control, CTS control, and RTS control
	ARM_USART_ERROR_BAUDRATE Baud rate setting error If the specified baud rate cannot be realized, a baud rate setting error occurs.
	ARM_USART_ERROR_MODE Mode setting error If ARM_USART_MODE_SINGLE_WIRE or ARM_USART_MODE_IRDA is specified as a control command, a mode setting error occurs.

Remarks	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	sci0Drv->Control(ARM_USART_CONTROL_TX, 1);
	}

Table 4-13 Behaviors Specified with Control Commands and Command-Specific Arguments

Control Command (control)	Command-Specific Argument (arg)	Description
ARM_USART_MODE_ASYNCHRONOUS	Baud rate (1 to 4294967295)	Initializes asynchronous mode with the specified baud rate.
ARM_USART_MODE_SYNCHRONOUS_MASTER	Baud rate (1 to 4294967295)	Initializes clock synchronous master mode with the specified baud rate.
ARM_USART_MODE_SYNCHRONOUS_SLAVE	NULL(0)	No arguments are used.
ARM_USART_MODE_SMART_CARD	Baud rate (1 to 4294967295)	Initializes smart card mode with the specified baud rate.
ARM_USART_SET_DEFAULT_TX_VALUE	Default data value (0x00 to 0xFF)	Sets transmit data to be output during reception in clock synchronous mode.
ARM_USART_SET_SMART_CARD_CLOCK	Output baud rate (1 to 4294967295)	Enables smart card clock output (only if the current baud rate matches the output baud rate)
	0	Disables smart card clock output.
ARM_USART_CONTROL_SMART_CARD_NACK	1	Enables NACK output in smart card mode (default: enabled)
ARM_USART_CONTROL_TX	1	Sets transmission enabled state.
	0	Sets transmission disabled state.
ARM_USART_CONTROL_RX	1	Sets reception enabled state.
	0	Sets reception disabled state.
ARM_USART_CONTROL_TX_RX	1	Sets transmission and reception enabled state.
	0	Sets transmission and reception disabled state.
ARM_USART_ABORT_SEND	NULL(0)	No arguments are used.
ARM_USART_ABORT_RECEIVE	NULL(0)	No arguments are used.
ARM_USART_ABORT_TRANSFER	NULL(0)	No arguments are used.

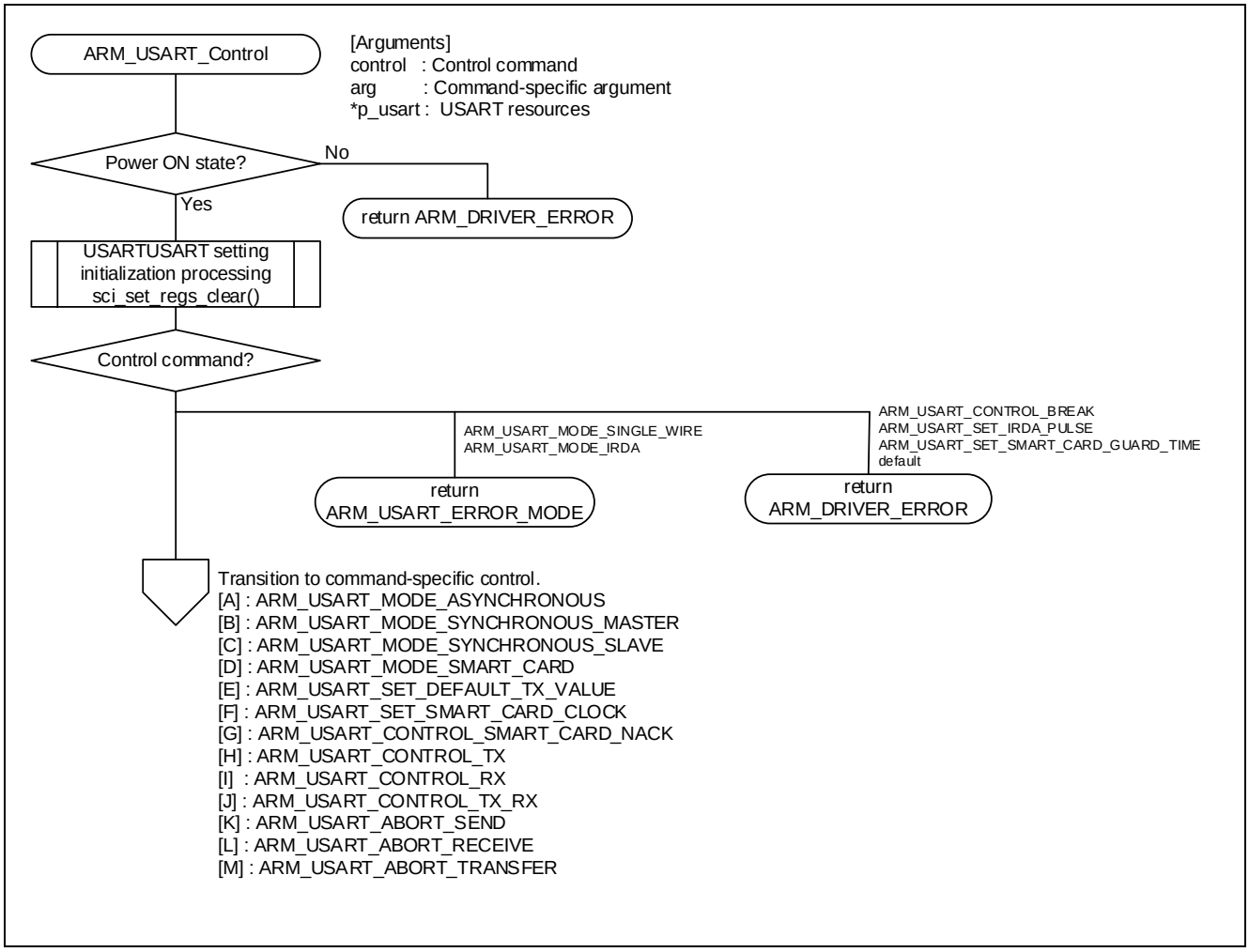


Figure 4-20 ARM_USART_Control Function Processing Flow (1/8)

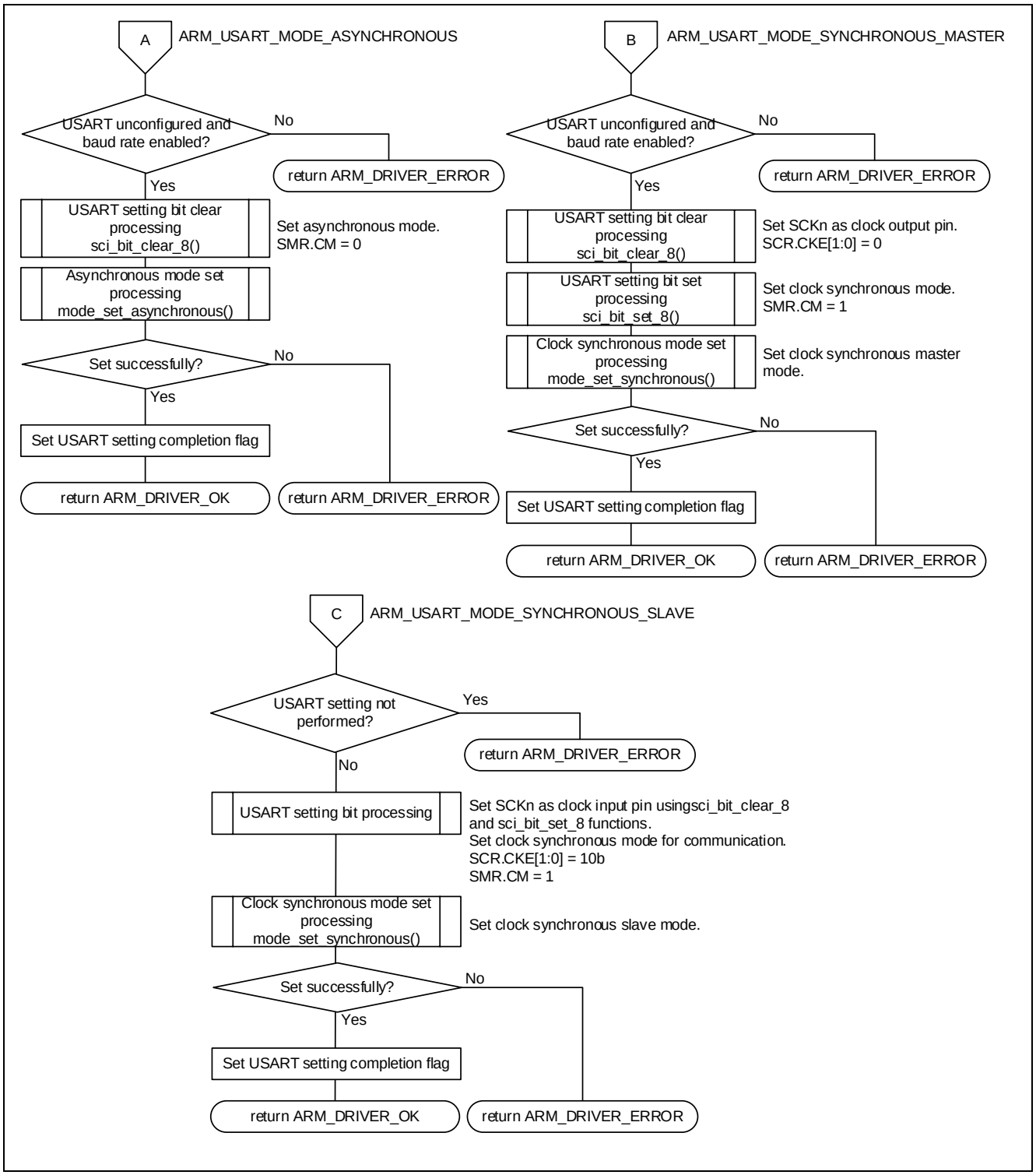


Figure 4-21 ARM_USART_Control Function Processing Flow (2/8)

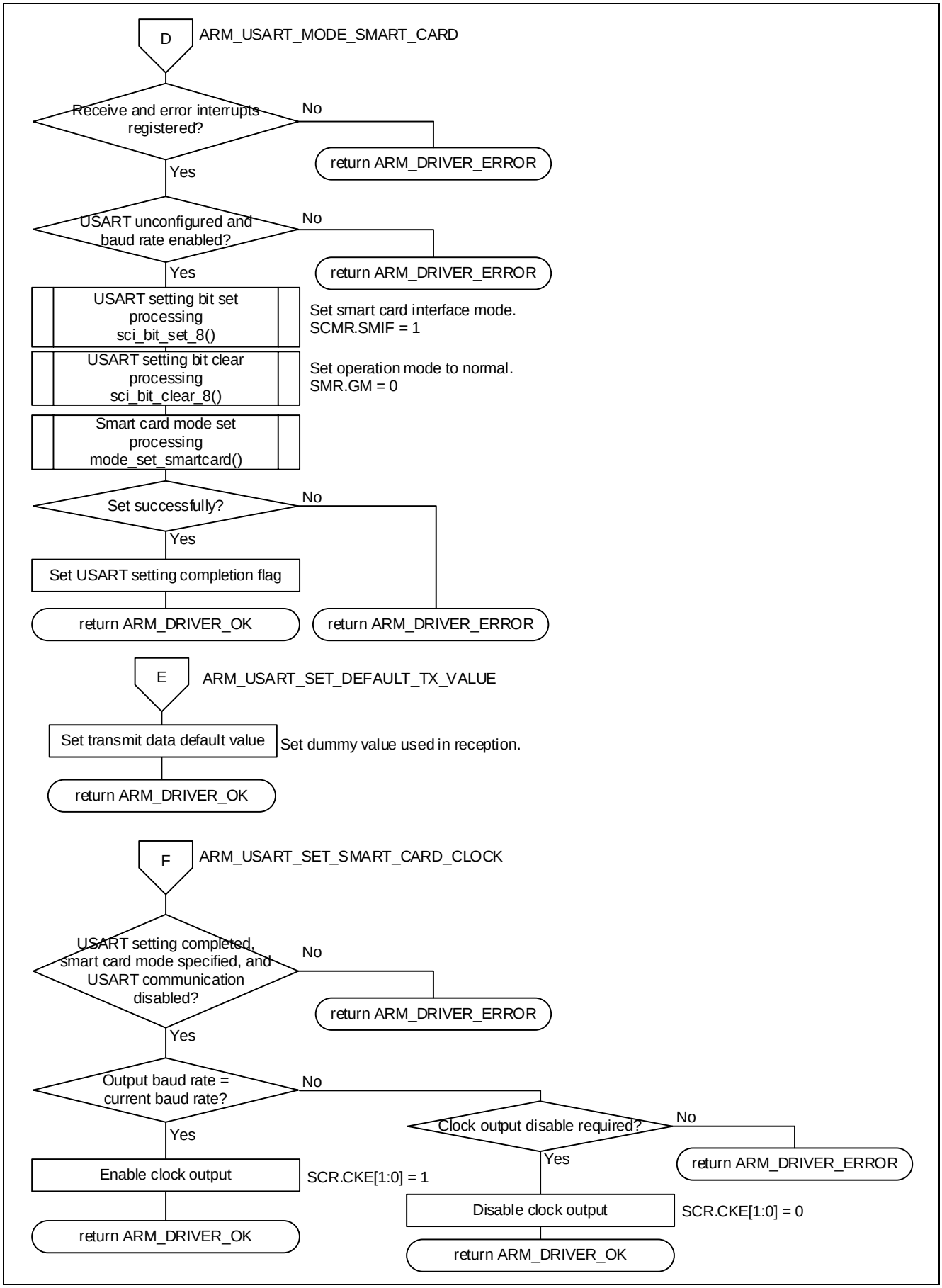


Figure 4-22 ARM_USART_Control Function Processing Flow (3/8)

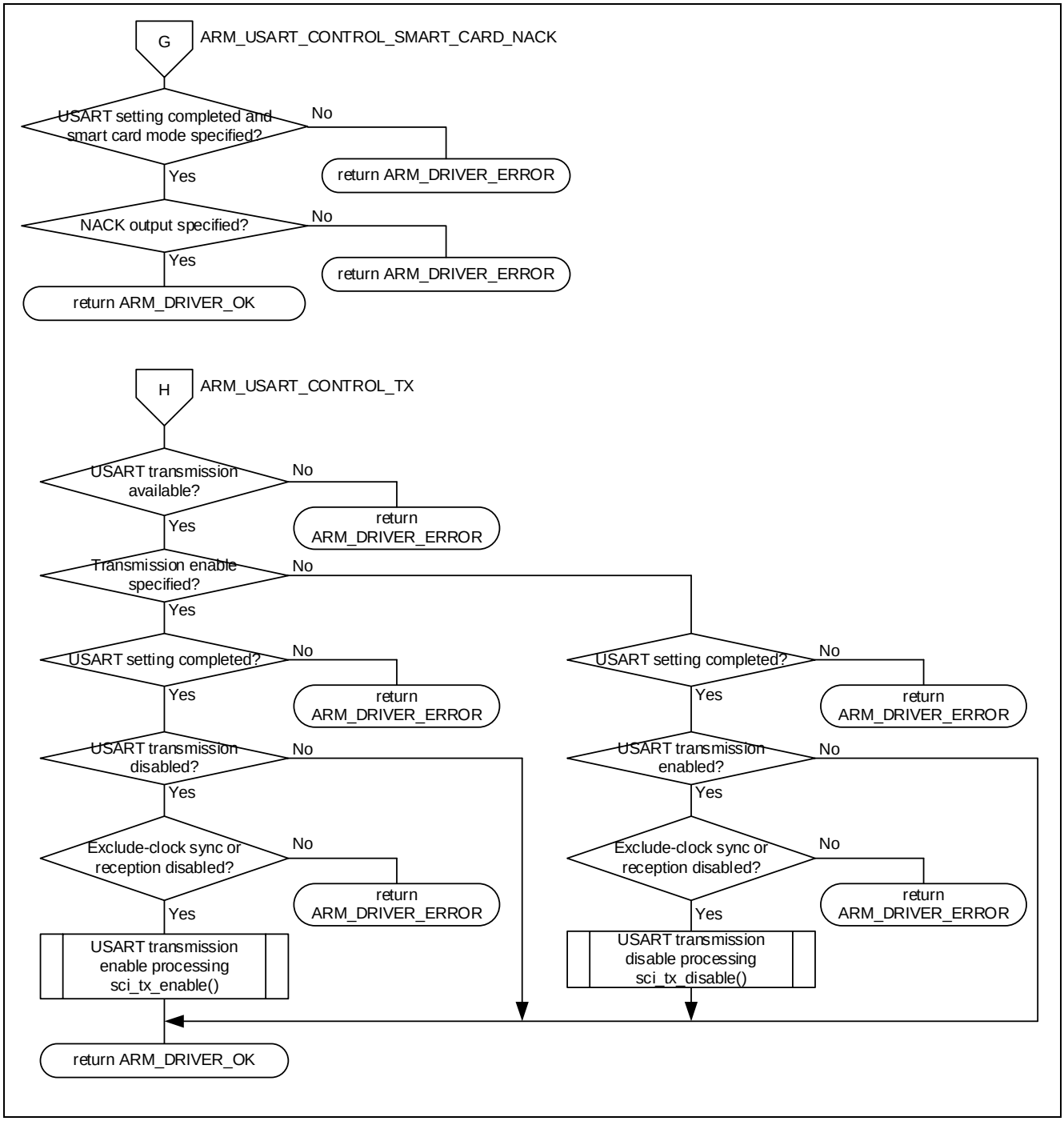


Figure 4-23 ARM_USART_Control Function Processing Flow (4/8)

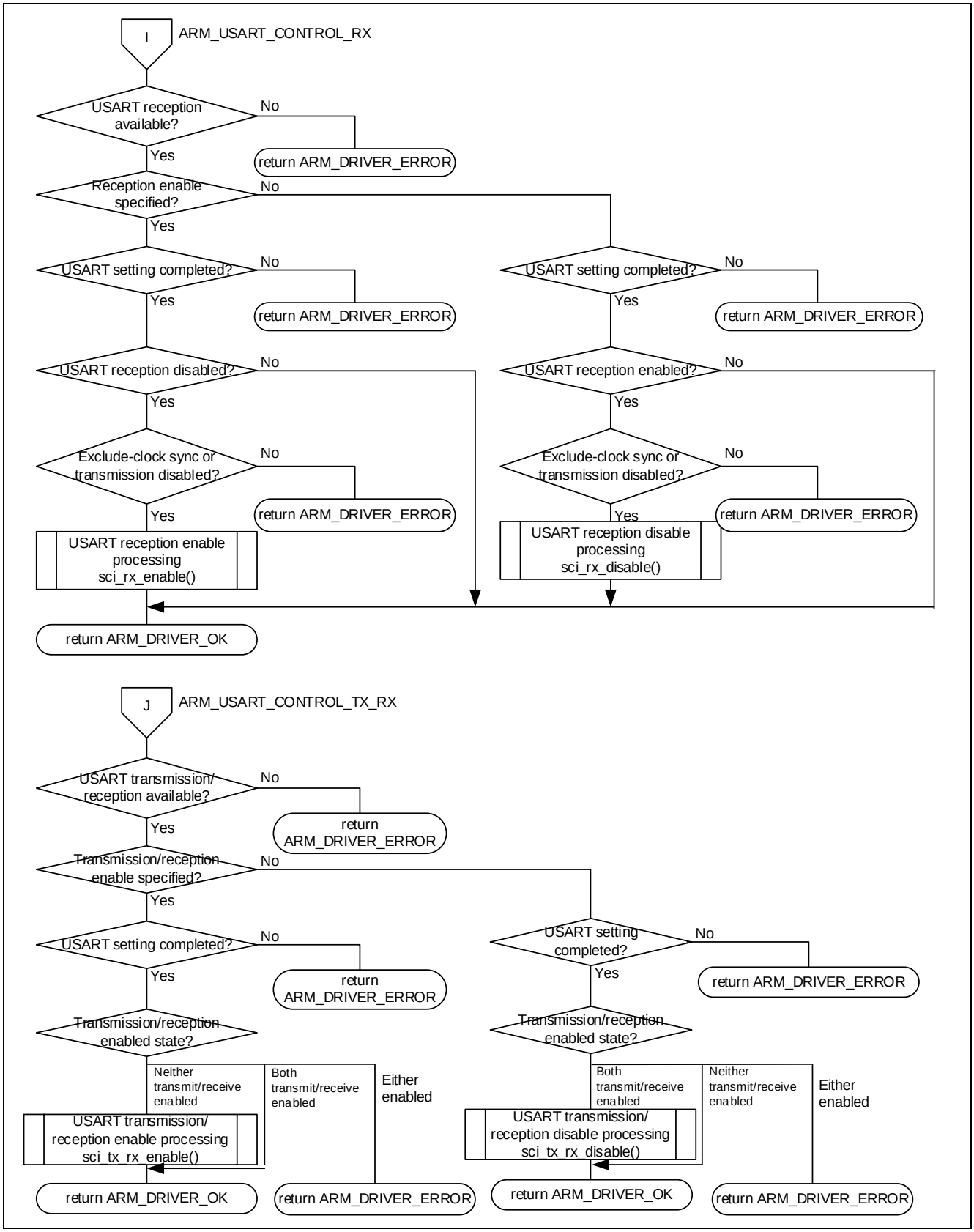


Figure 4-24 ARM_USART_Control Function Processing Flow (5/8)

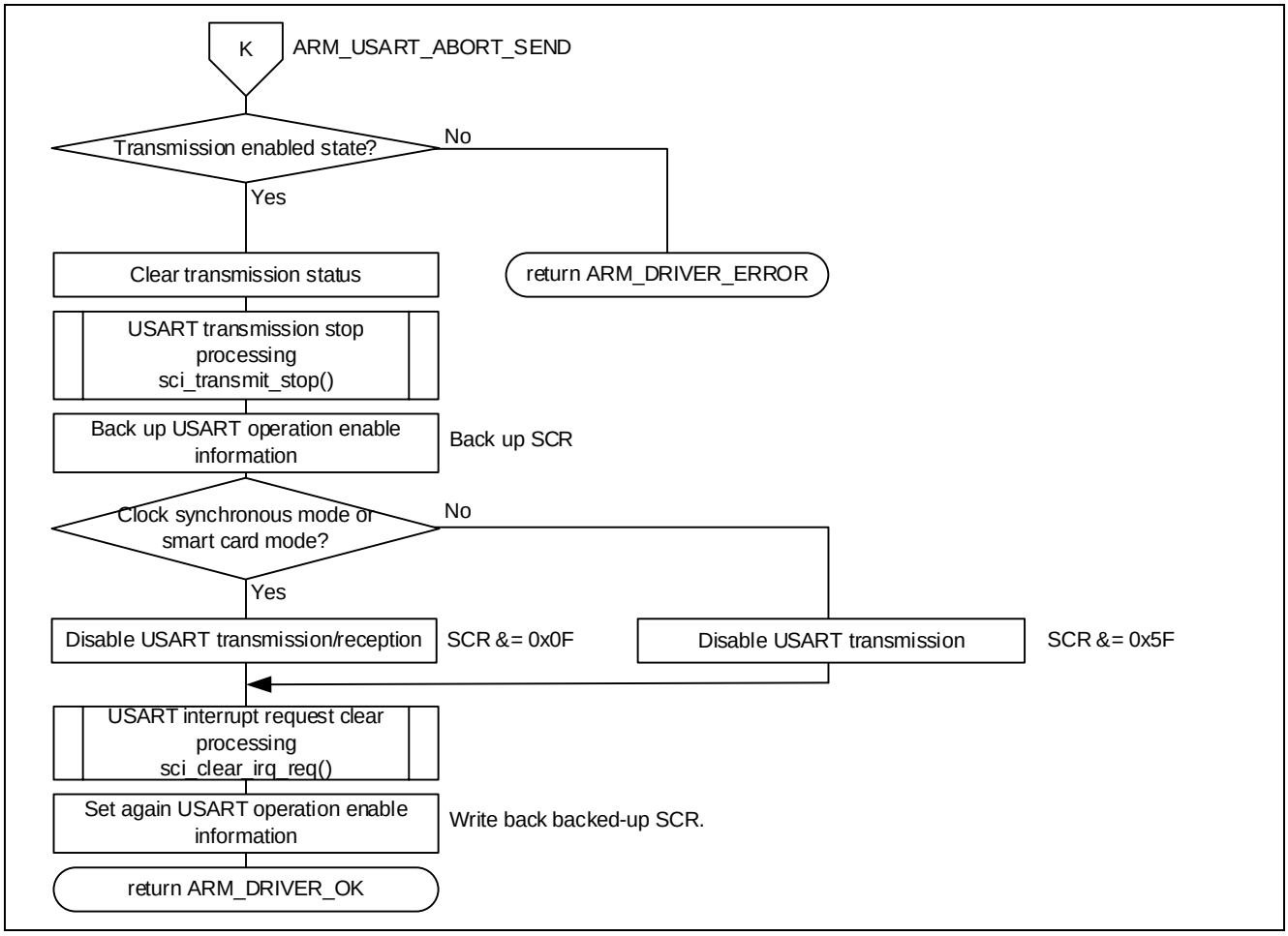


Figure 4-25 ARM_USART_Control Function Processing Flow (6/8)

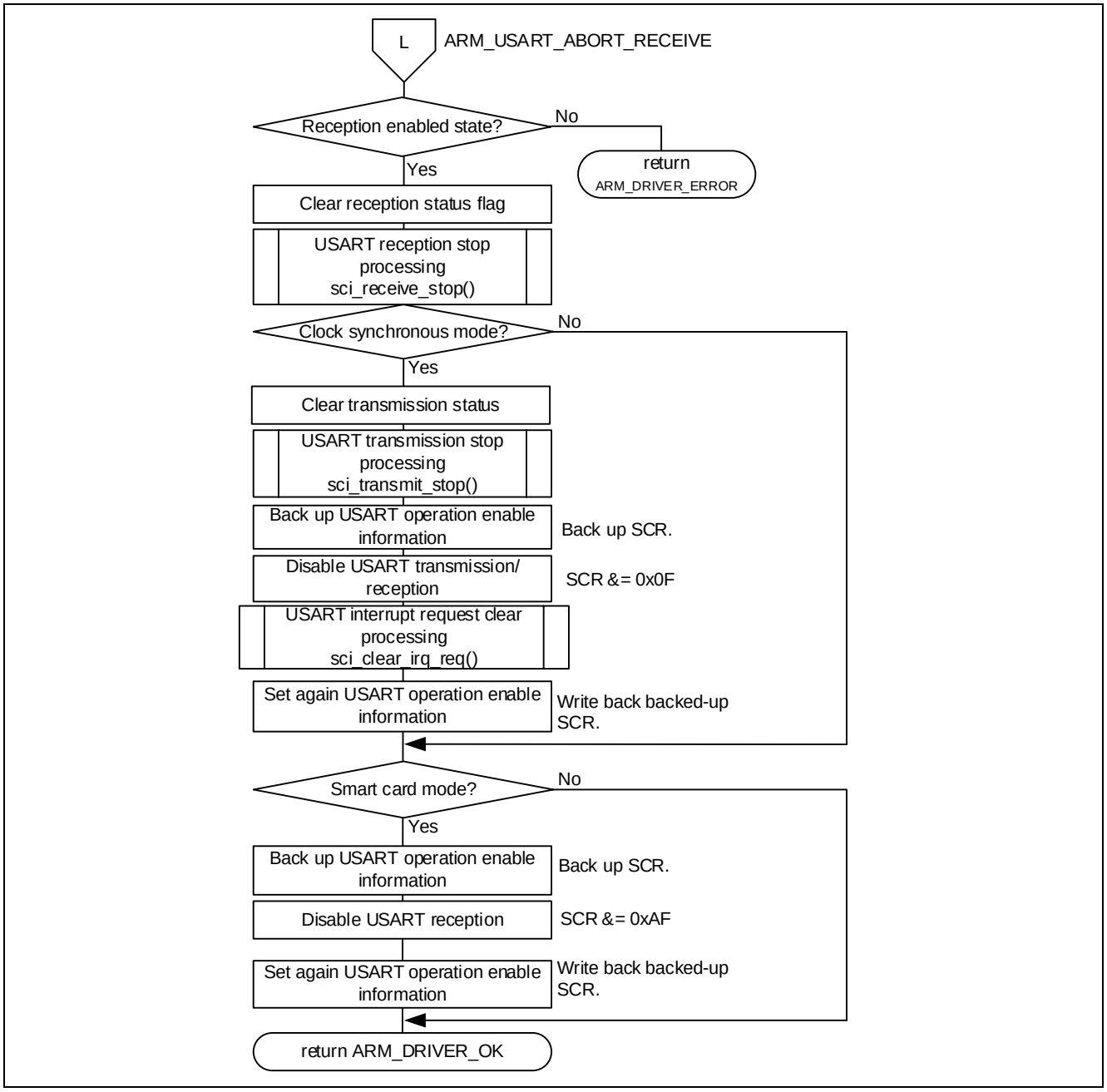


Figure 4-26 ARM_USART_Control Function Processing Flow (7/8)

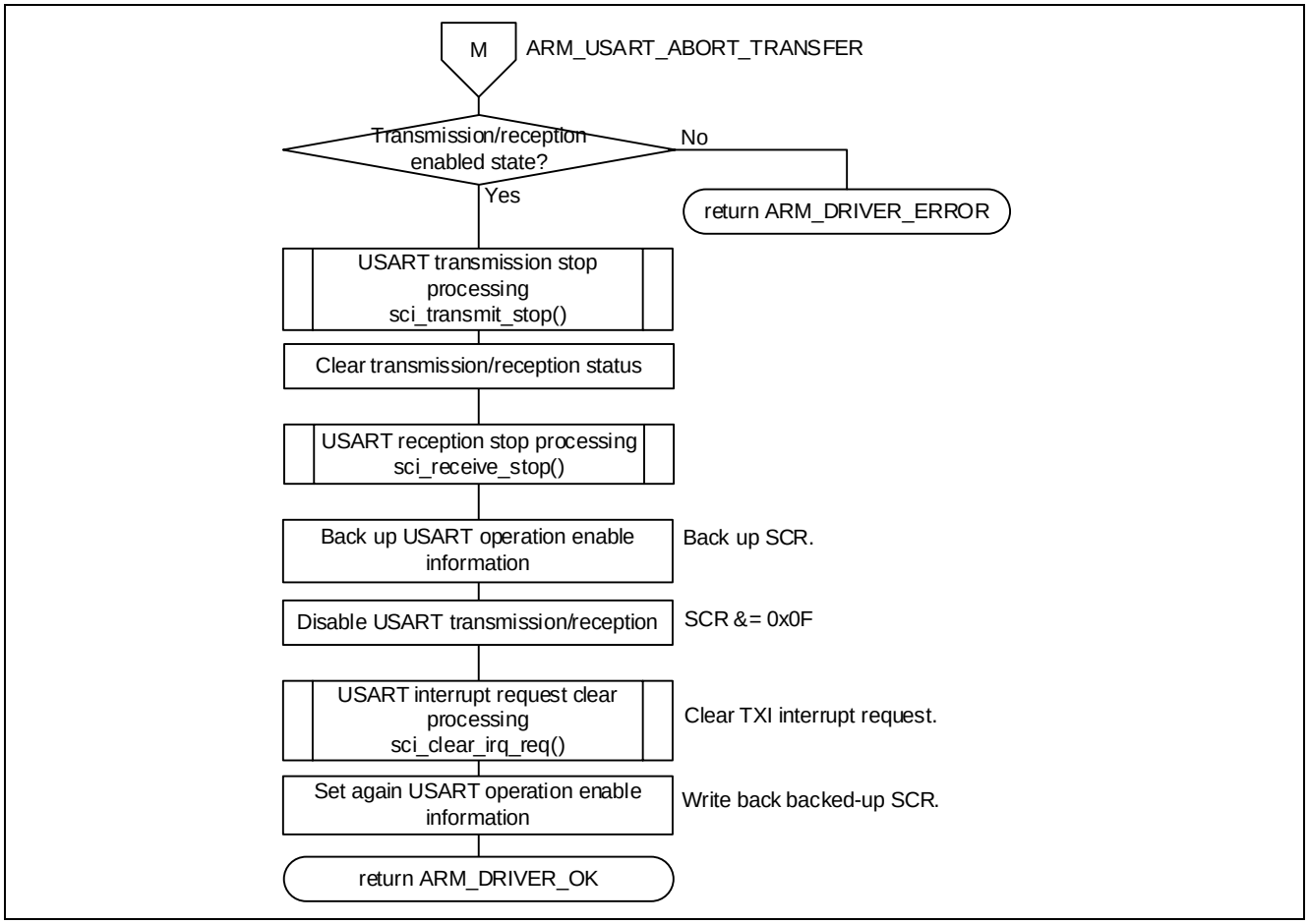


Figure 4-27 ARM_USART_Control Function Processing Flow (8/8)

4.1.12 ARM_USART_GetStatus Function

Table 4-14 ARM_USART_GetStatus Function Specifications

Format	ARM_USART_STATUS ARM_USART_GetStatus(st_usart_resources_t const * const p_usart)
Description	Returns the status of USART
Argument	st_usart_resources_t * const p_usart : Resources of USART Specifies the resources of the USART concerned.
Return value	Communication status

Remarks	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	ARM_USART_STATUS state;
	state = sci0Drv->GetStatus();
	
	}

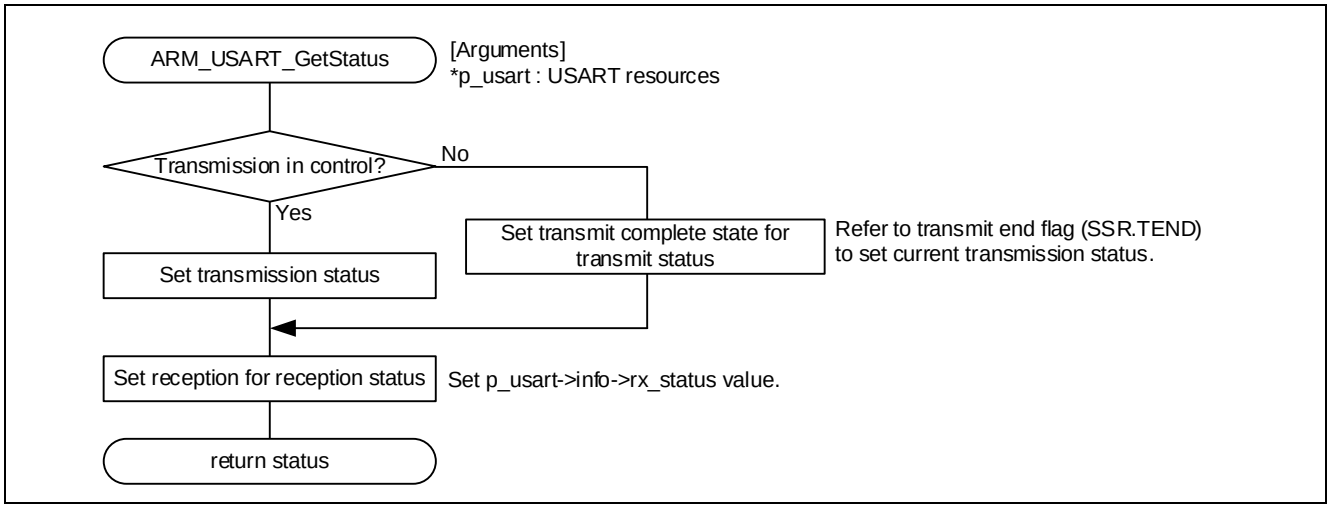


Figure 4-28 ARM_USART_GetStatus Function Processing Flow

4.1.13 ARM_USART_SetModemControl Function

Table 4-15 ARM_USART_SetModemControl Function Specifications

Format	int32_t ARM_USART_SetModemControl(ARM_USART_MODEM_CONTROL control, st_usart_resources_t const * const p_usart)
Description	Performs modem control with software.
Argument	ARM_USART_MODEM_CONTROL control : Modem control commands See section 2.5.3 Modem Control Definition for the modem control commands.
	st_usart_resources_t * const p_usart : Resources of USART Specifies USART resources to be controlled.
Return value	ARM_DRIVER_OK Modem control performed normally
	ARM_DRIVER_ERROR Modem control failed If one of the following conditions is detected, the transmission/reception start will fail. • If this function is executed without setting RTS pin by r_usart_cfg.h • If this function is executed before setting USART • If this function is executed during operation with RTS control • If an illegal command is set for a modem control command

Remarks	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	sci0Drv->SetModemControl(ARM_USART_RTS_SET);
	
	}

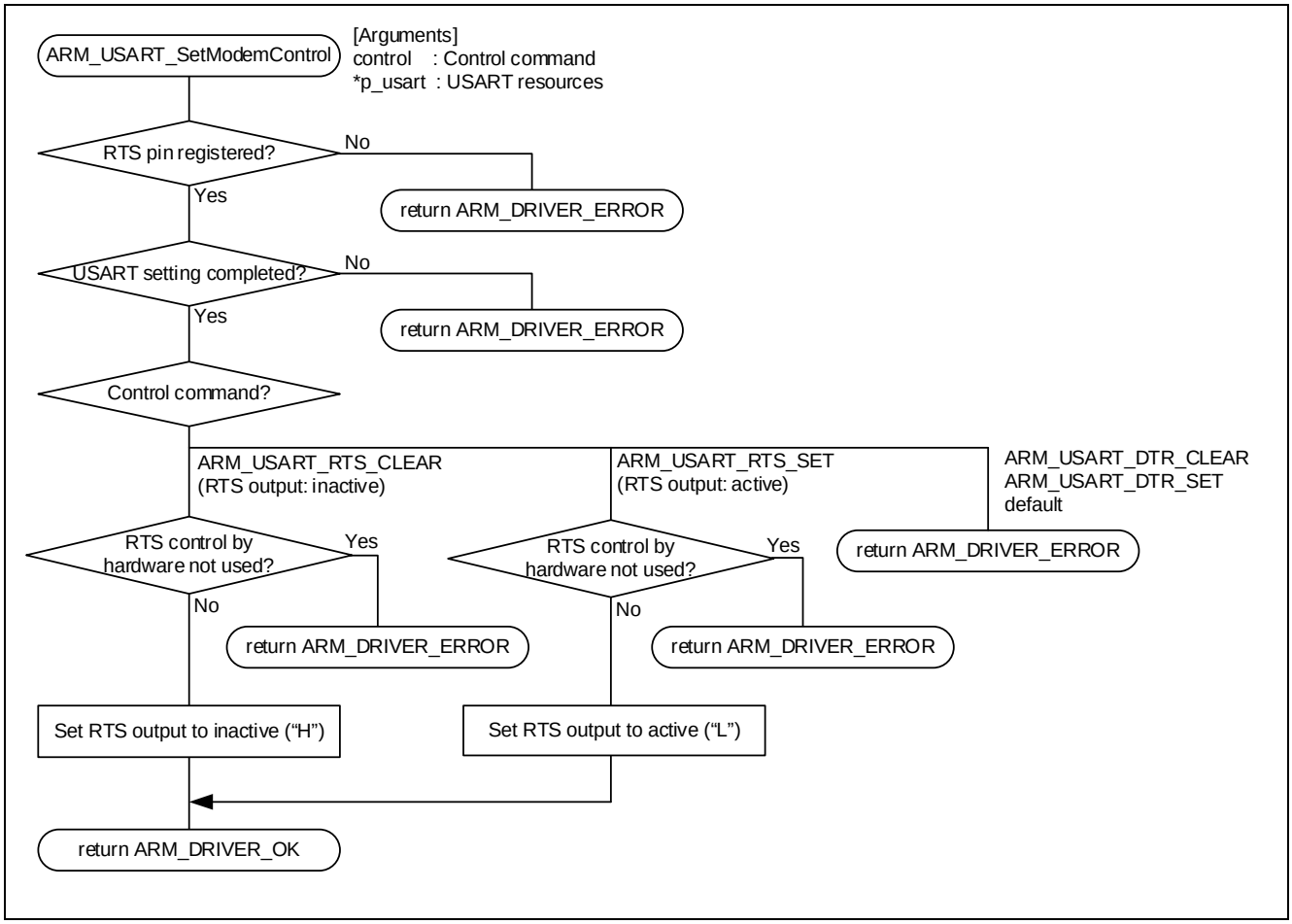


Figure 4-29 ARM_USART_SetModemControl Function Processing Flow

4.1.14 ARM_USART_GetModemStatus Function

Table 4-16 ARM_USART_GetModemStatus Function Specifications

Format	ARM_USART_MODEM_STATUS ARM_USART_GetModemStatus(st_usart_resources_t const * const p_usart)
Description	Acquires CTS pin status.
Argument	st_usart_resources_t * const p_usart : Resources of USART Specifies USART resources to be controlled.
Return value	Modem status

Remarks	This function is enabled only when CTS control with hardware is not used.
	When this function is accessed, specifying the USART resources is not required.
	
	[Example of calling function from instance]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	ARM_USART_MODEM_STATUS modem_state;
	modem_state = sci0Drv->GetModemStatus();
	
	}

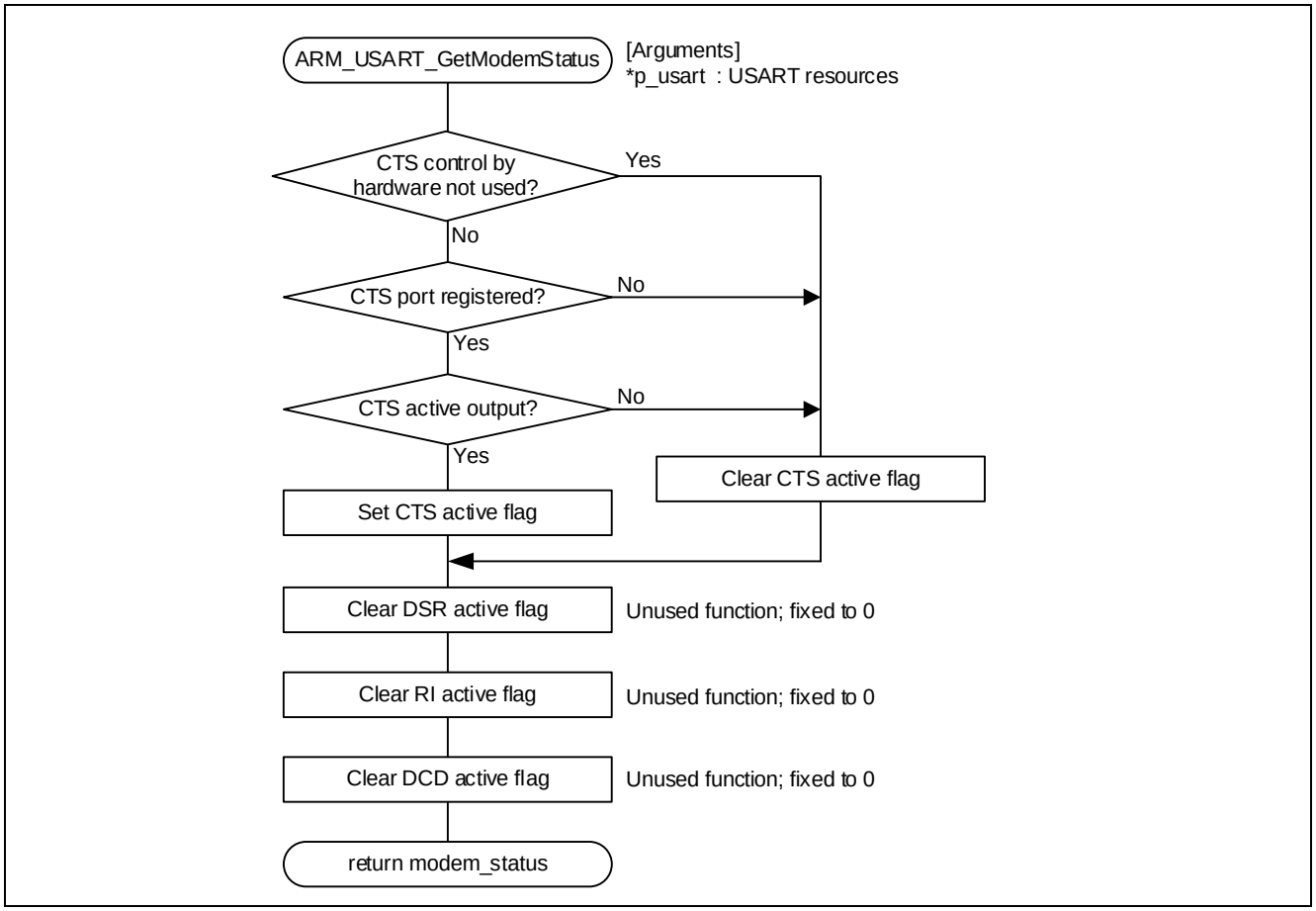


Figure 4-30 ARM_USART_GetModemStatus Function Processing Flow

4.1.15 mode_set_asynchronous Function

Table 4-17 mode_set_asynchronous Function Specifications

Format	static int32_t mode_set_asynchronous(uint32_t control, st_sci_reg_set_t * const p_sci_regs, uint32_t baud, st_usart_resources_t * const p_usart)	
Description	Specifies asynchronous mode settings.	
Argument	uint32_t control: Control command	
	st_sci_reg_set_t * const p_sci_regs: Pointer to register setting value storage	
	uint32_t baud: Baud rate setting	
	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.	
Return value	ARM_DRIVER_OK	Asynchronous mode set normally
	ARM_USART_ERROR_DATA_BITS	Data bit length setting error If a data bit length other than 7, 8, or 9 bits is specified, a data bit length setting error will occur.
	ARM_USART_ERROR_PARITY	Parity setting error If a parity setting other than no parity, odd parity, or even parity is specified, a parity setting error will occur.
	ARM_USART_ERROR_STOP_BITS	Stop bit length setting error If a stop bit length other than 1 or 2 bits is specified, a stop bit length setting error will occur.
	ARM_USART_ERROR_FLOW_CONTROL	Flow control setting error If a flow control other than no flow control, CTS control, or RTS control is specified, a flow control setting error will occur.
	ARM_USART_ERROR_BAUDRATE	Baud rate setting error If the specified baud rate cannot be realized, a baud rate setting error will occur.
Remarks		

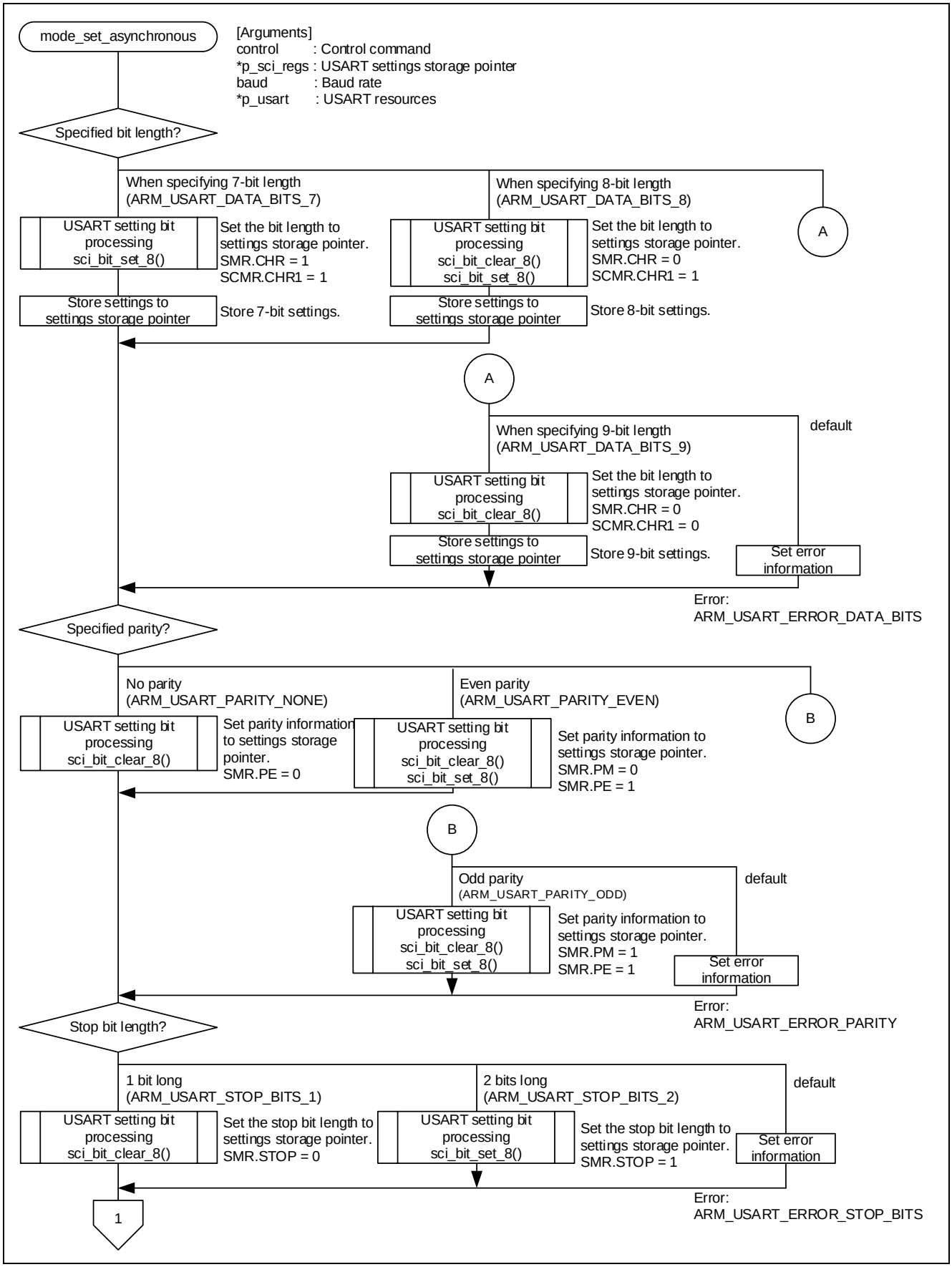


Figure 4-31 mode_set_asynchronous Function Processing Flow (1/2)

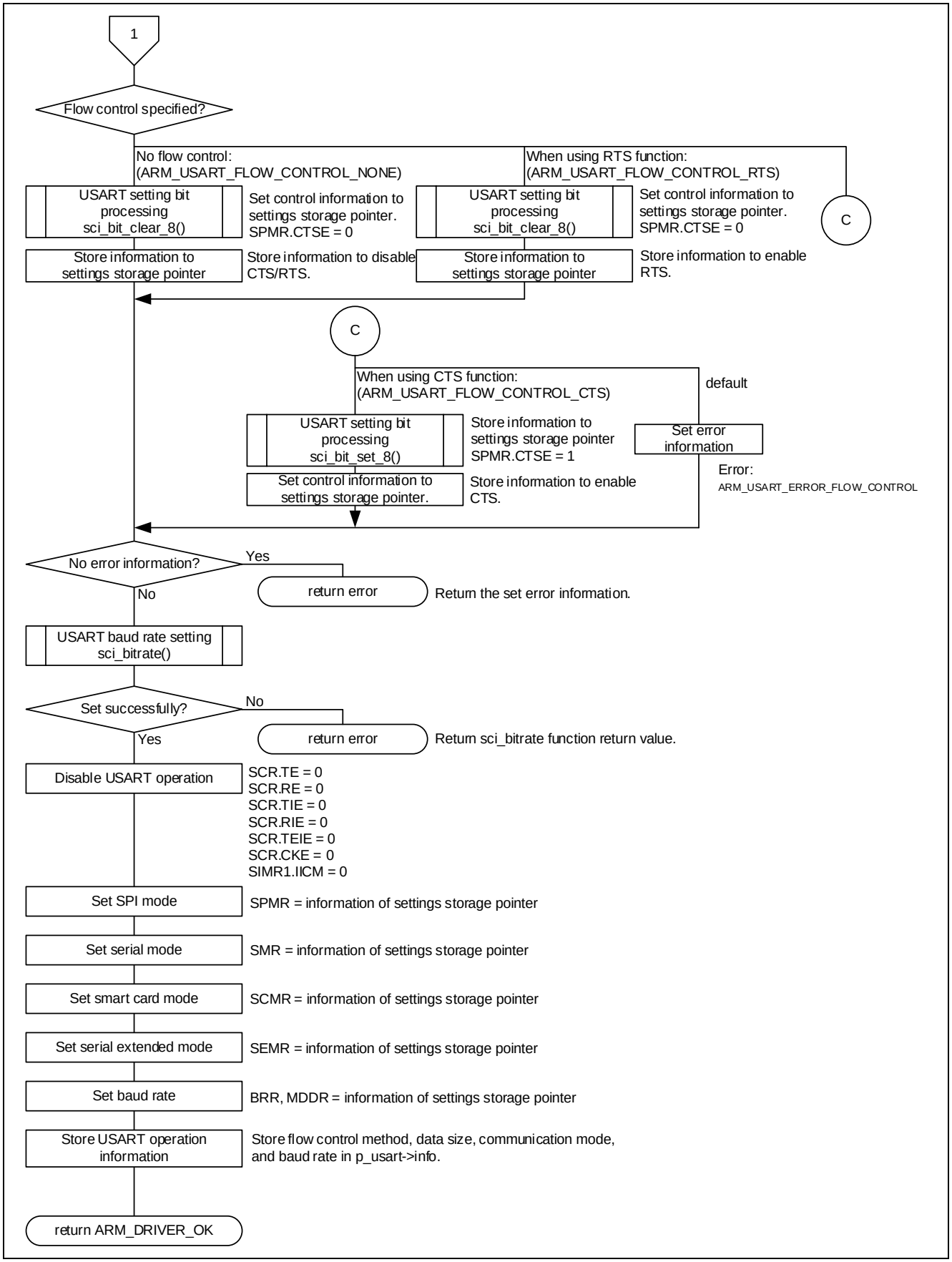


Figure 4-32 mode_set_asynchronous Function Processing Flow (2/2)

4.1.16 mode_set_synchronous Function

Table 4-18 mode_set_synchronous Function Specifications

Format	static int32_t mode_set_synchronous(uint32_t control, st_sci_reg_set_t * const p_sci_regs, uint32_t baud, st_usart_resources_t * const p_usart)
Description	Specifies clock synchronous mode settings.
Argument	uint32_t control: Control command
	st_sci_reg_set_t * const p_sci_regs: Pointer to register setting value storage
	uint32_t baud: Baud rate setting
	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.
Return value	ARM_DRIVER_OK Clock synchronous mode set normally
	ARM_USART_ERROR_FLOW_CONTROL Flow control setting error If a flow control other than no flow control, CTS control, and RTS control is specified, a flow control setting error will occur.
	ARM_USART_ERROR_BAUDRATE Baud rate setting error If the specified baud rate cannot be realized, a baud rate setting error will occur.
Remarks	

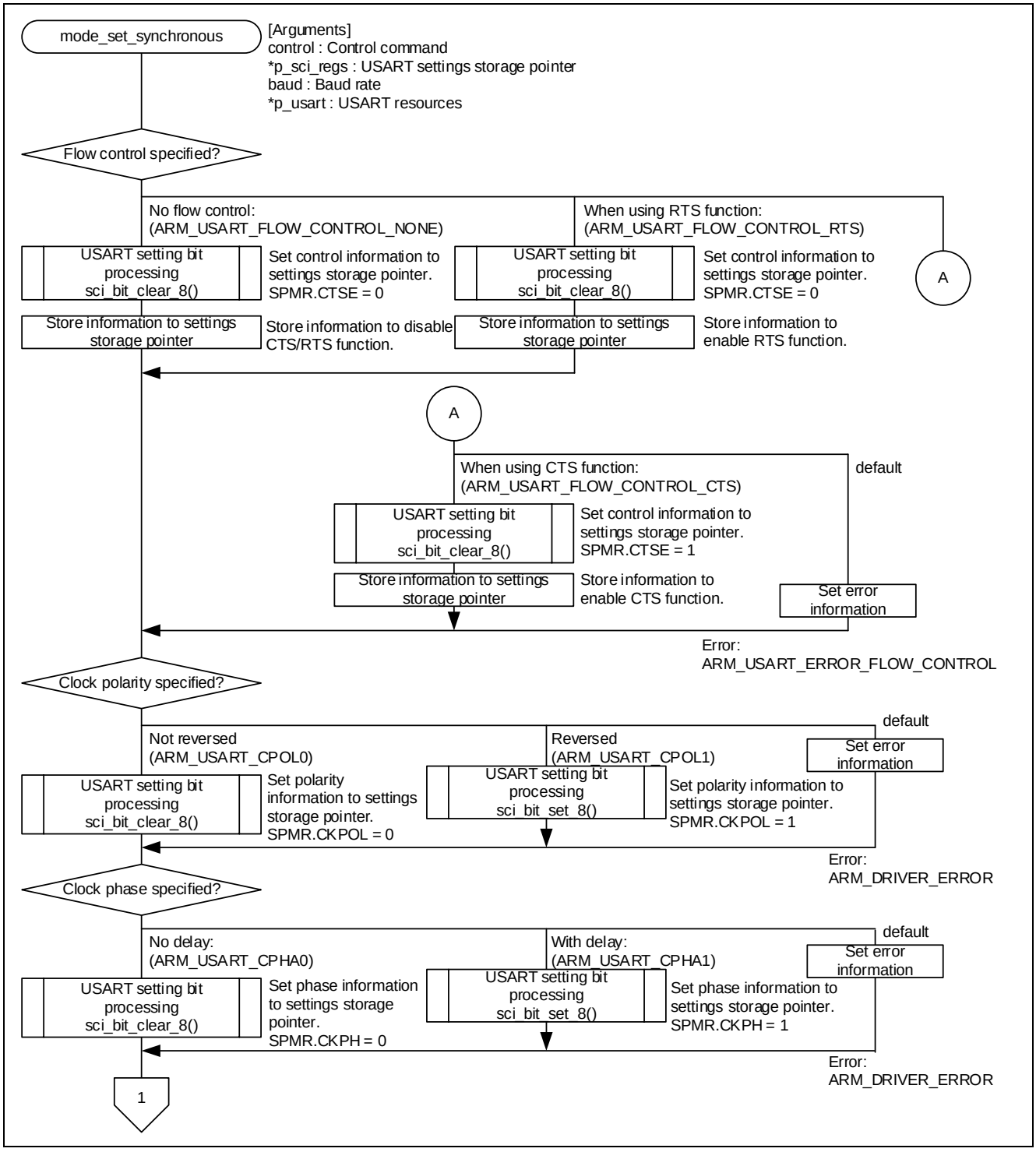


Figure 4-33 mode_set_synchronous Function Processing Flow (1/2)

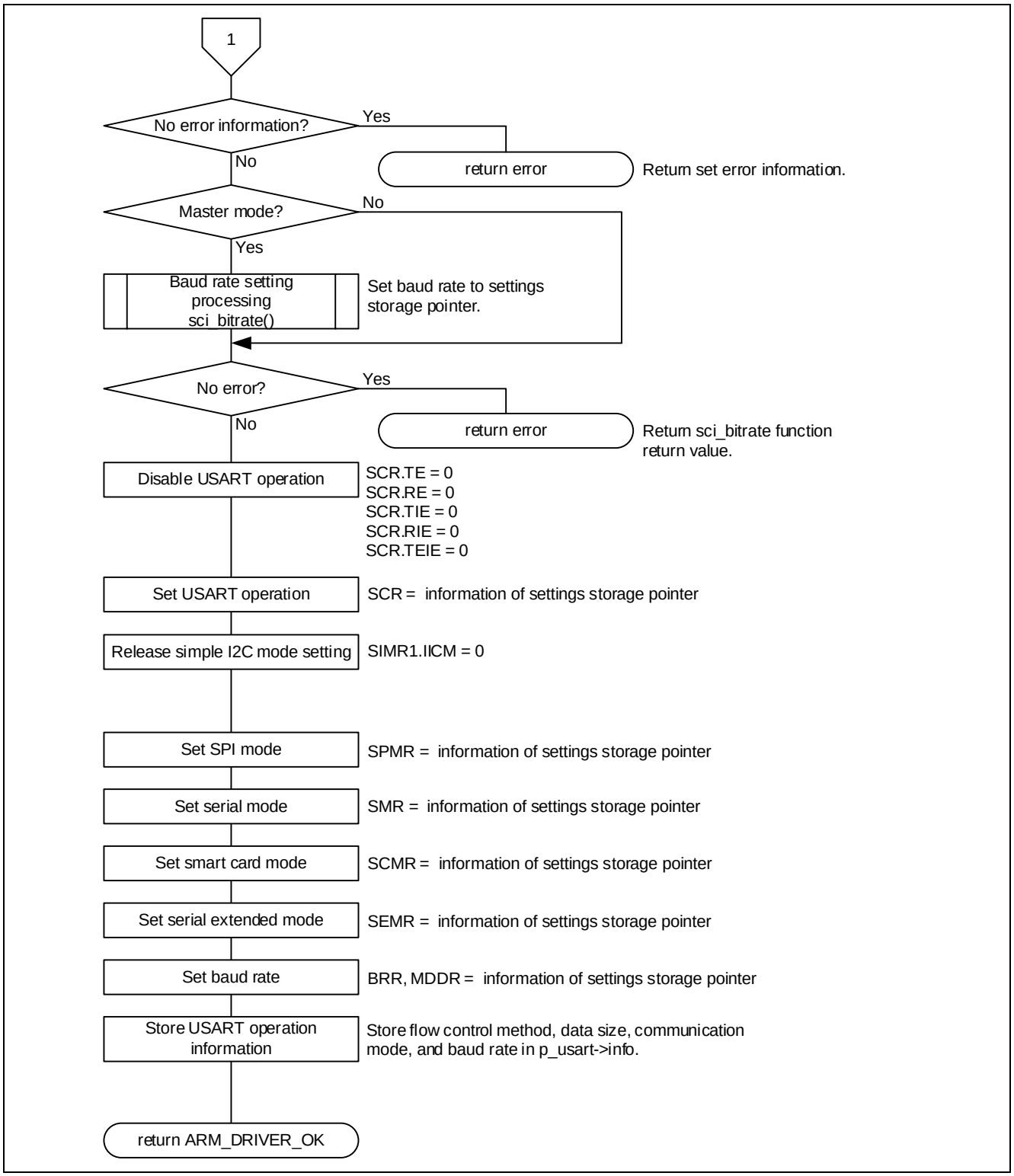


Figure 4-34 mode_set_synchronous Function Processing Flow (2/2)

4.1.17 mode_set_smartcard Function

Table 4-19 mode_set_smartcard Function Specifications

Format	static int32_t mode_set_smartcard(uint32_t control, st_sci_reg_set_t * const p_sci_regs, uint32_t baud, st_usart_resources_t * const p_usart)	
Description	Specifies smart card mode settings.	
Argument	uint32_t control : Control command	
	st_sci_reg_set_t * const p_sci_regs : Pointer to register setting value storage	
	uint32_t baud : Baud rate setting	
	st_usart_resources_t * const p_usart : Resources of USART Specifies USART resources to be controlled.	
Return value	ARM_DRIVER_OK	Smart card mode set normally
	ARM_USART_ERROR_PARITY	Parity setting error If a parity other than no parity, odd parity, and even parity is specified, a parity setting error will occur.
	ARM_USART_ERROR_BAUDRATE	Baud rate setting error If the specified baud rate cannot be realized, a baud rate setting error will occur.
Remarks		

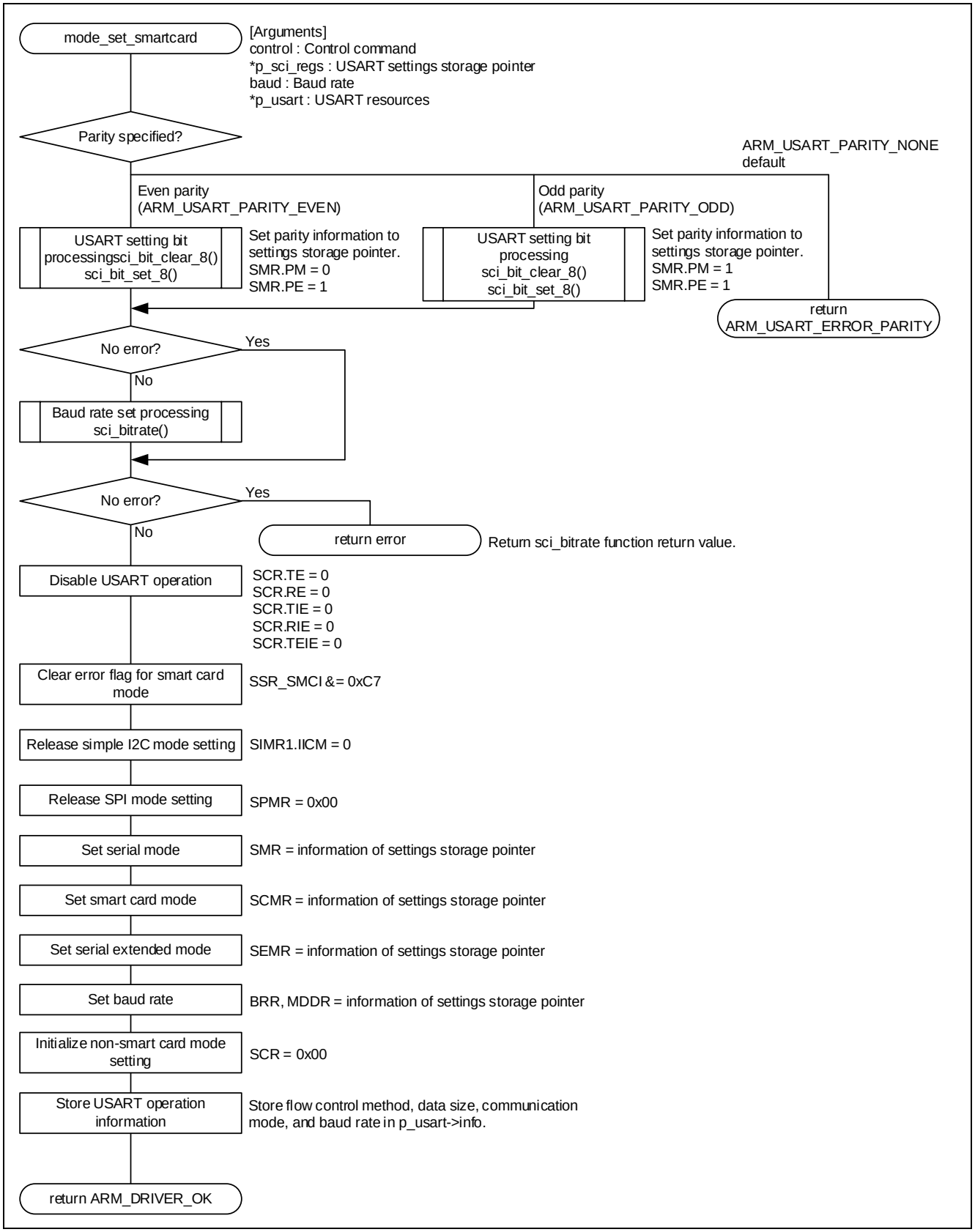
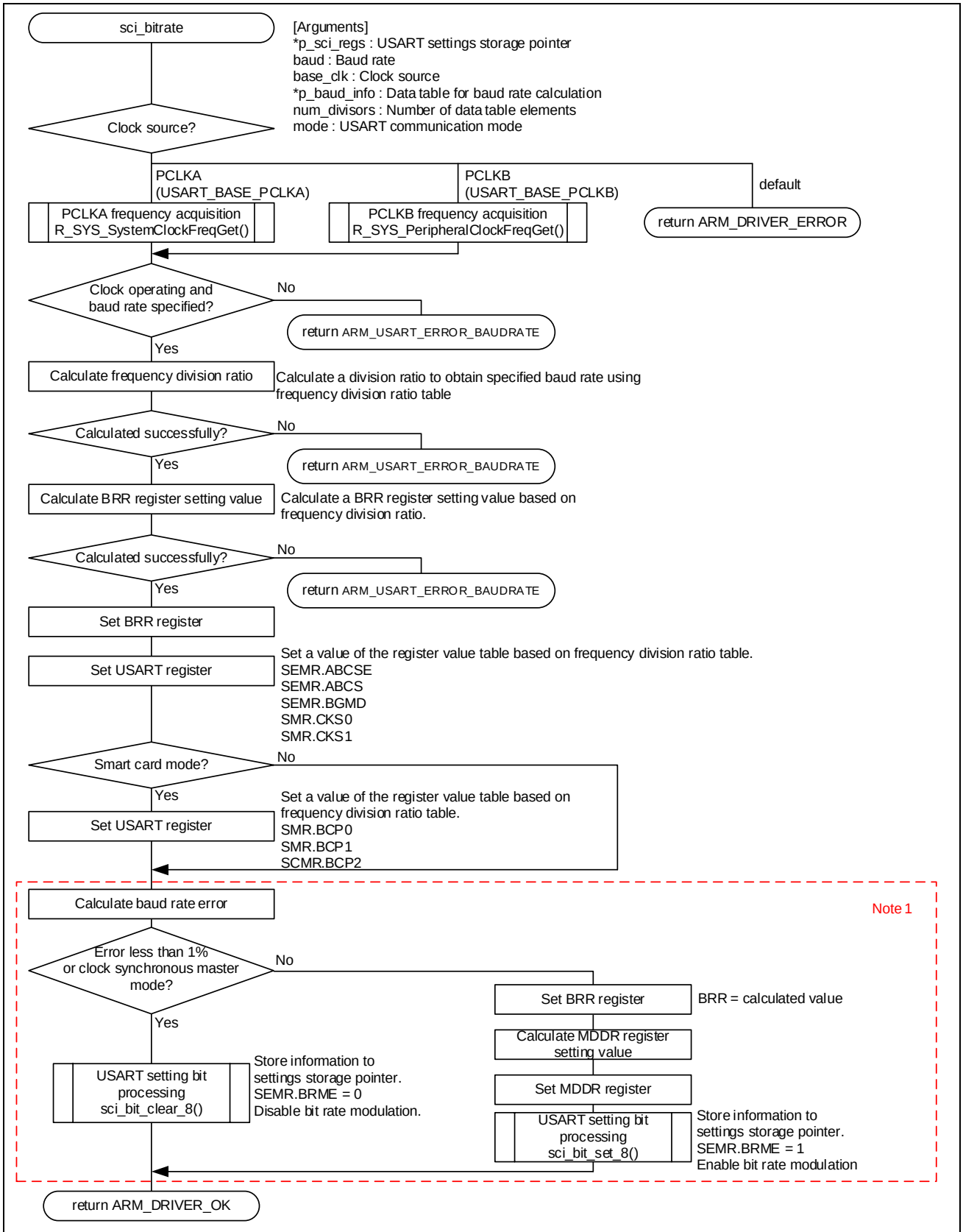


Figure 4-35 mode_set_smartcard Function Processing Flow

4.1.18 sci_bitrate Function

Table 4-20 sci_bitrate Function Specifications

Format	static int32_t sci_bitrate(st_sci_reg_set_t *p_sci_regs, uint32_t baud, e_usart_base_clk_t base_clk, st_baud_divisor_t const *p_baud_info, uint8_t num_divisors, uint32_t mode)	
Description	Calculates the baud rate.	
Argument	st_sci_reg_set_t *p_sci_regs: Pointer to register setting buffer Pointer to buffer for storing bus speed calculation results	
	uint32_t baud: Bus speed	
	e_usart_base_clk_t base_clk: Base clock	
	st_baud_divisor_t const *p_baud_info: Table of frequency division ratios	
	uint8_t num_divisors: Size of the frequency division table	
	uint32_t mode: Operation mode	
Return value	ARM_DRIVER_OK	Bus speed calculation completed
	ARM_DRIVER_ERROR	Bus speed calculation failed If the selected base clock is not PCLKA or PCLKB, a bus speed calculation error will occur.
	ARM_USART_ERROR_BAUDRATE	Baud rate calculation failed
Remarks		



Note 1. When bit rate modulation function is set to use with r_usart_cfg.h, the appropriate processing is enabled.

Figure 4-36 sci_bitrate Function Processing Flow

4.1.19 sci_set_regs_clear Function

Table 4-21 sci_set_regs_clear Function Specifications

Format	static void sci_set_regs_clear(st_sci_reg_set_t * const p_regs)
Description	Initializes the register setting buffer.
Argument	st_sci_reg_set_t *p_sci_regs: Pointer to the register setting buffer
Return value	None
Remarks	-

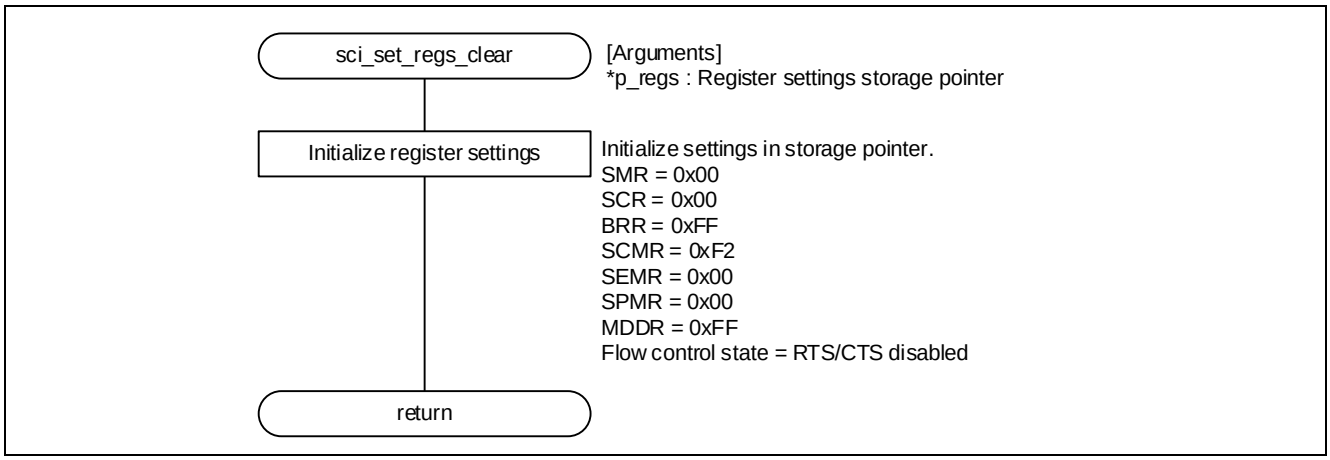


Figure 4-37 sci_set_regs_clear Function Processing Flow

4.1.20 sci_tx_enable Function

Table 4-22 sci_tx_enable Function Specifications

Format	static void sci_tx_enable(st_usart_resources_t * const p_usart)
Description	Enables transmission.
Argument	st_usart_resources_t * const p_usart : Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	

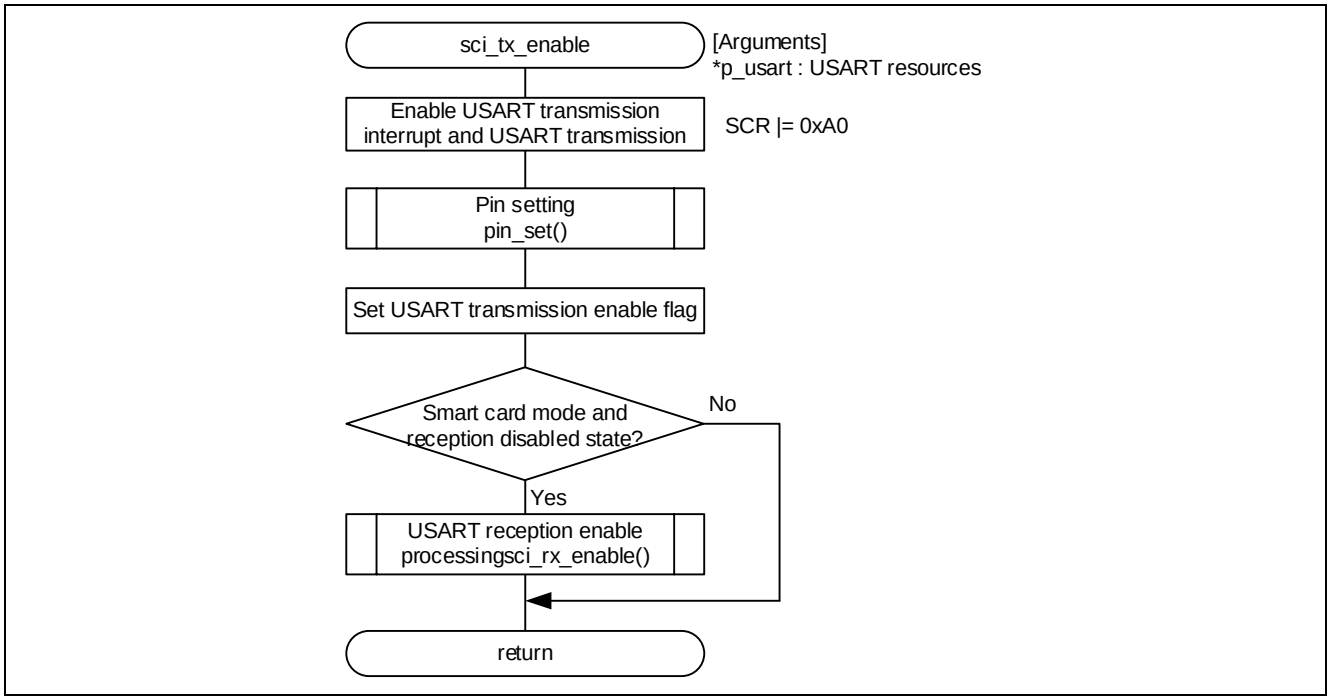


Figure 4-38 sci_tx_enable Function Processing Flow

4.1.21 sci_tx_disable Function

Table 4-23 sci_tx_disable Function Specifications

Format	static void sci_tx_disable(st_usart_resources_t * const p_usart)
Description	Disables transmission.
Argument	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	–

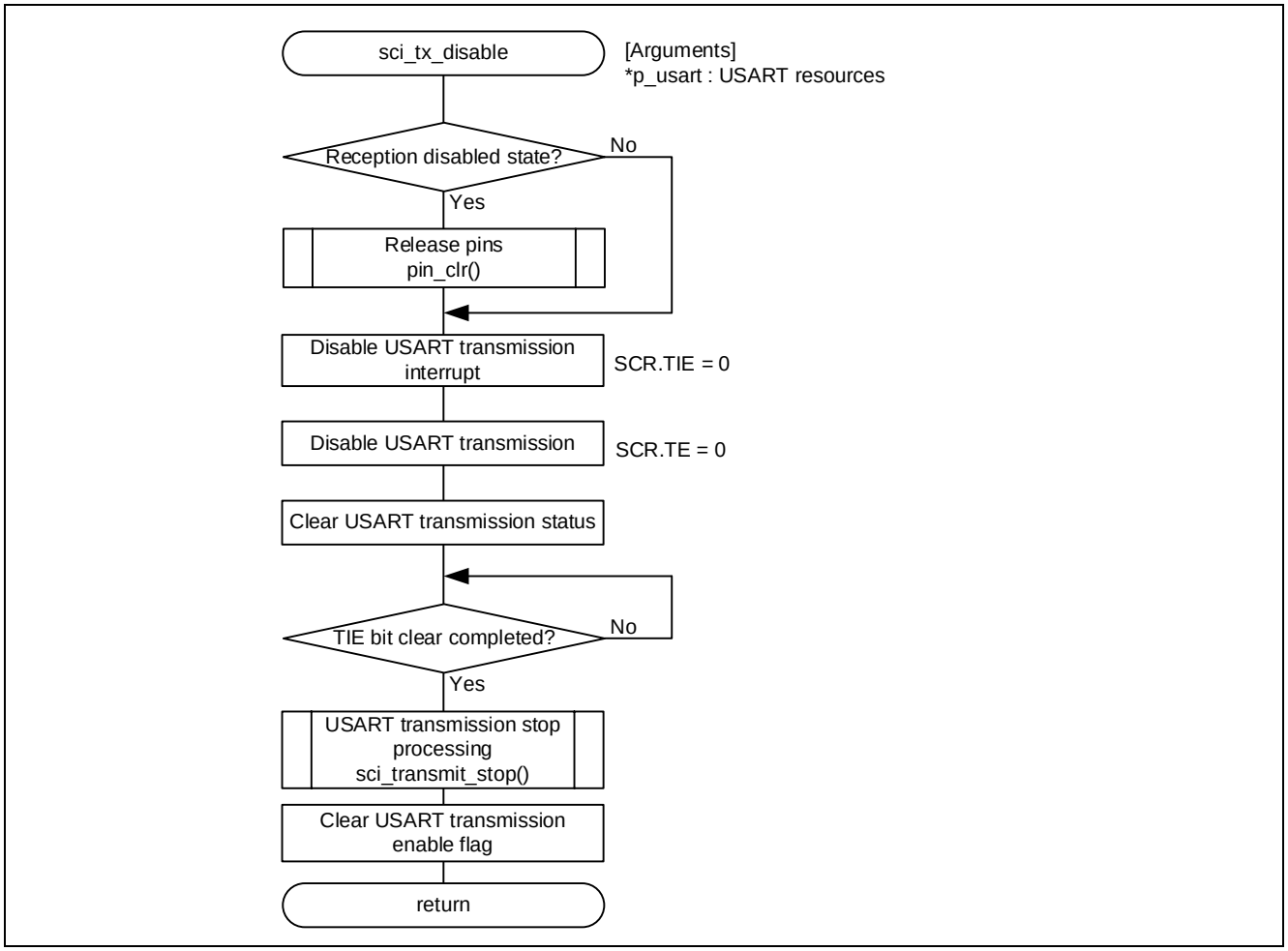


Figure 4-39 sci_tx_disable Function Processing Flow

4.1.22 sci_rx_enable Function

Table 4-24 sci_rx_enable Function Specifications

Format	static void sci_rx_enable(st_usart_resources_t * const p_usart)
Description	Enables reception.
Argument	st_usart_resources_t * const p_usart : Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	–

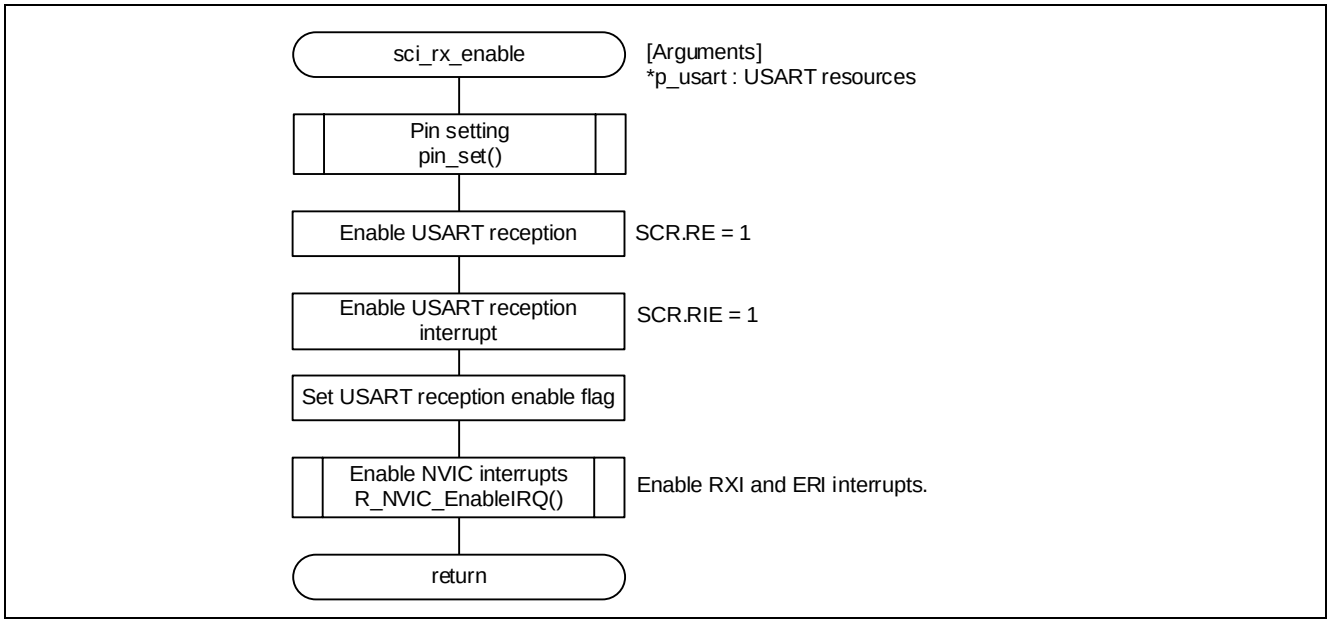


Figure 4-40 sci_rx_enable Function Processing Flow

4.1.23 sci_rx_disable Function

Table 4-25 sci_rx_disable Function Specifications

Format	static void sci_rx_disable(st_usart_resources_t * const p_usart)
Description	Disables reception.
Argument	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	–

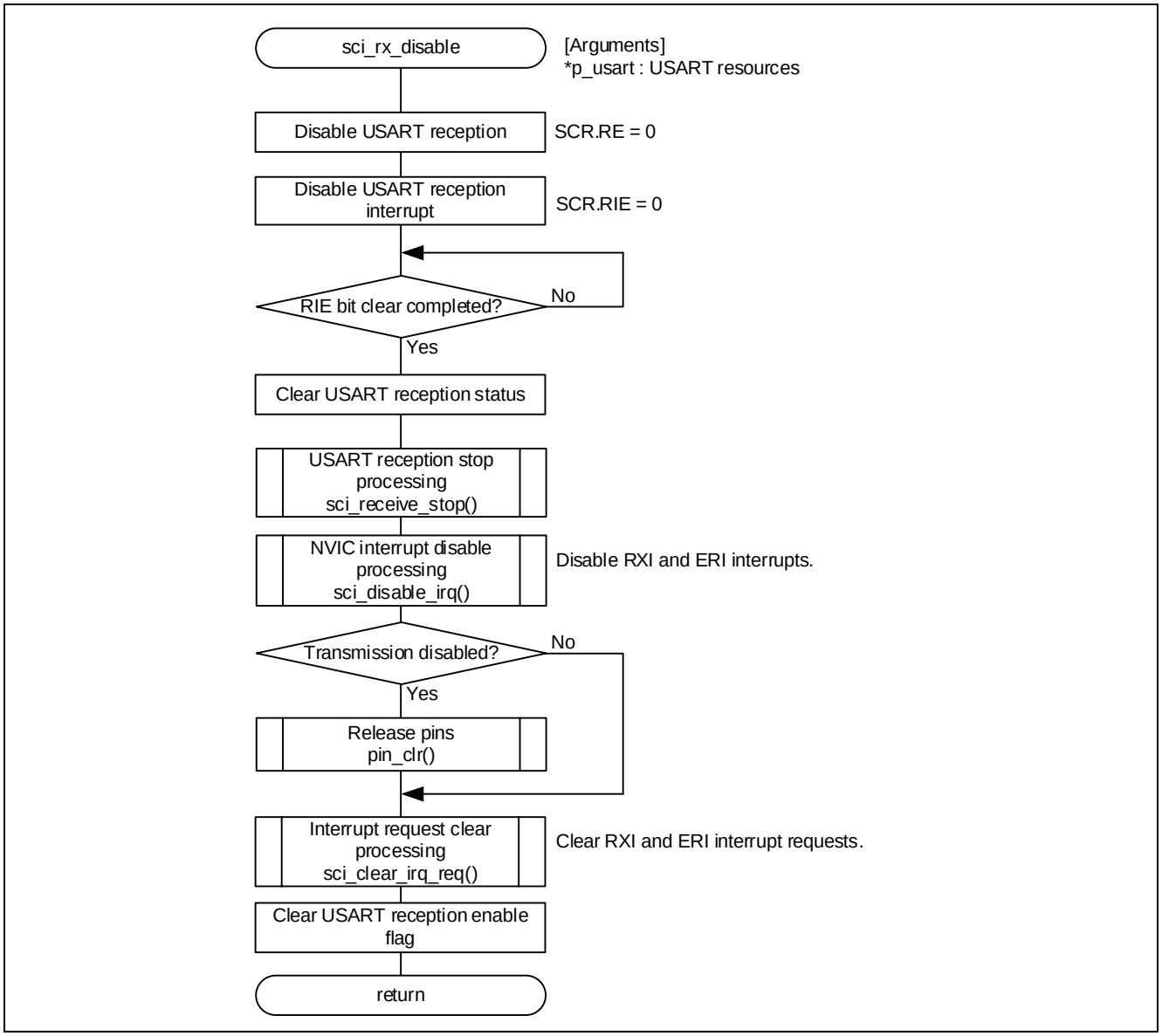


Figure 4-41 sci_rx_disable Function Processing Flow

4.1.24 sci_tx_rx_enable Function

Table 4-26 sci_tx_rx_enable Function Specifications

Format	static void sci_tx_rx_enable (st_usart_resources_t * const p_usart)
Description	Enables transmission and reception.
Argument	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	

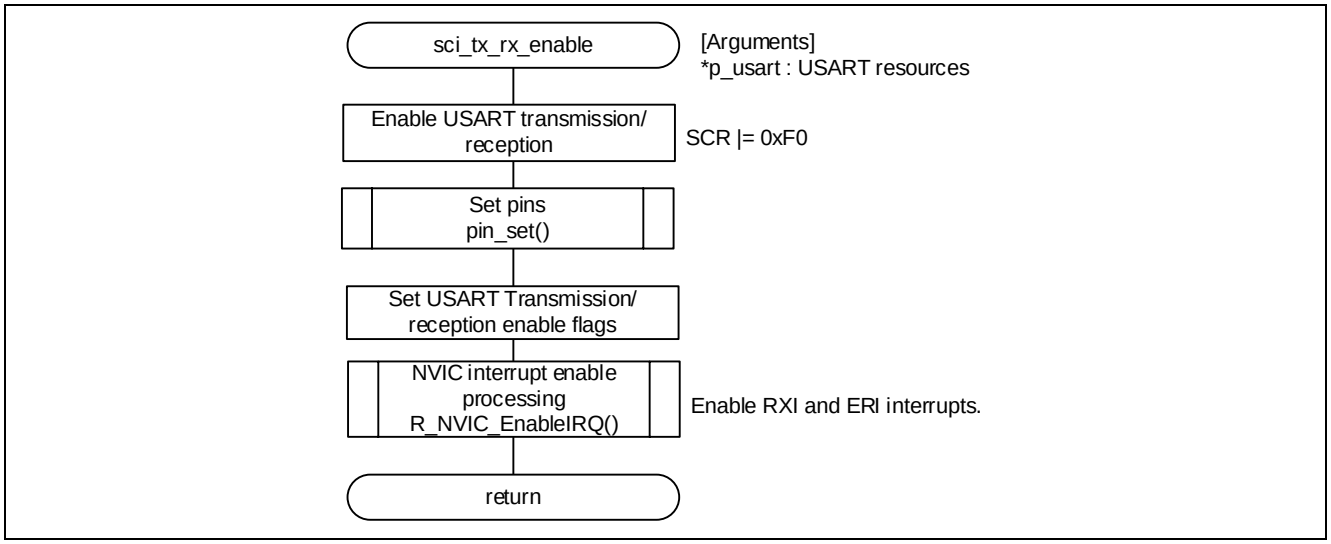


Figure 4-42 sci_tx_rx_enable Function Processing Flow

4.1.25 sci_tx_rx_disable Function

Table 4-27 sci_tx_rx_disable Function Specifications

Format	static void sci_tx_rx_disable (st_usart_resources_t * const p_usart)
Description	Disables transmission and reception.
Argument	st_usart_resources_t * const p_usart : Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	

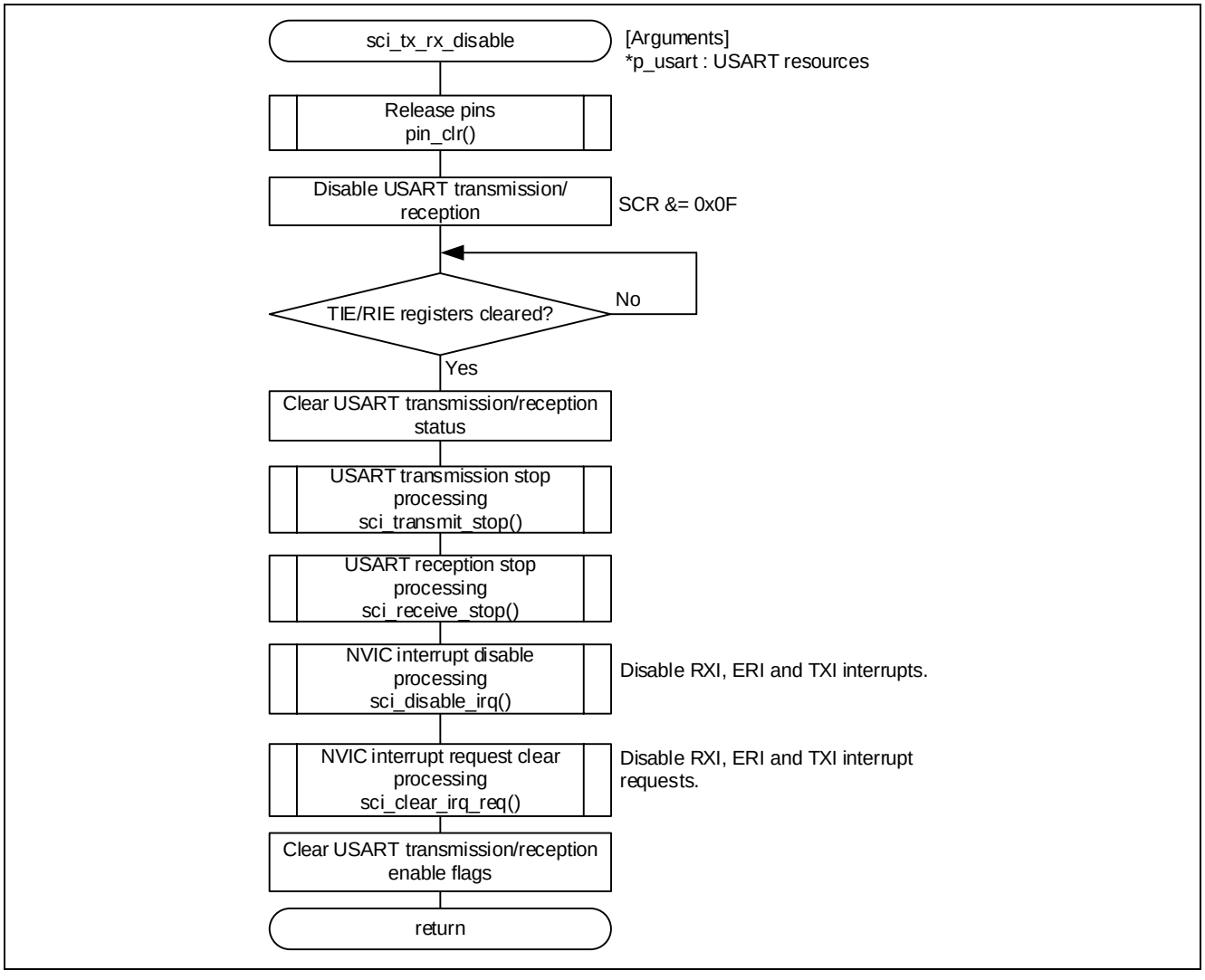


Figure 4-43 sci_tx_rx_disable Function Processing Flow

4.1.26 check_tx_available Function

Table 4-28 check_tx_available Function Specifications

Format	static int32_t check_tx_available(int16_t * const p_flag, st_usart_resources_t * const p_usart)
Description	Judges whether or not transmission is available.
Argument	int16_t * const p_flag: Pointer to initialization flag storage
	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.
Return value	ARM_DRIVER_OK Transmission availability judgment completed
	ARM_DRIVER_ERROR Transmission availability judgment failed If one of the following conditions is detected, the transmission availability judgment will fail. • If TXI interrupt event link setting fails when interrupts or DTC is used for transmission. • If interrupt priority level setting fails when interrupts or DTC is used for transmission. • If TXI interrupt is defined not to be used (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED) with r_system_cfg.h when DTC is used for transmission. • If DMA driver initialization fails when DTC or DMAC is used for transmission. • If DMAC interrupt enable setting fails when DMAC is used for transmission.
Remarks	

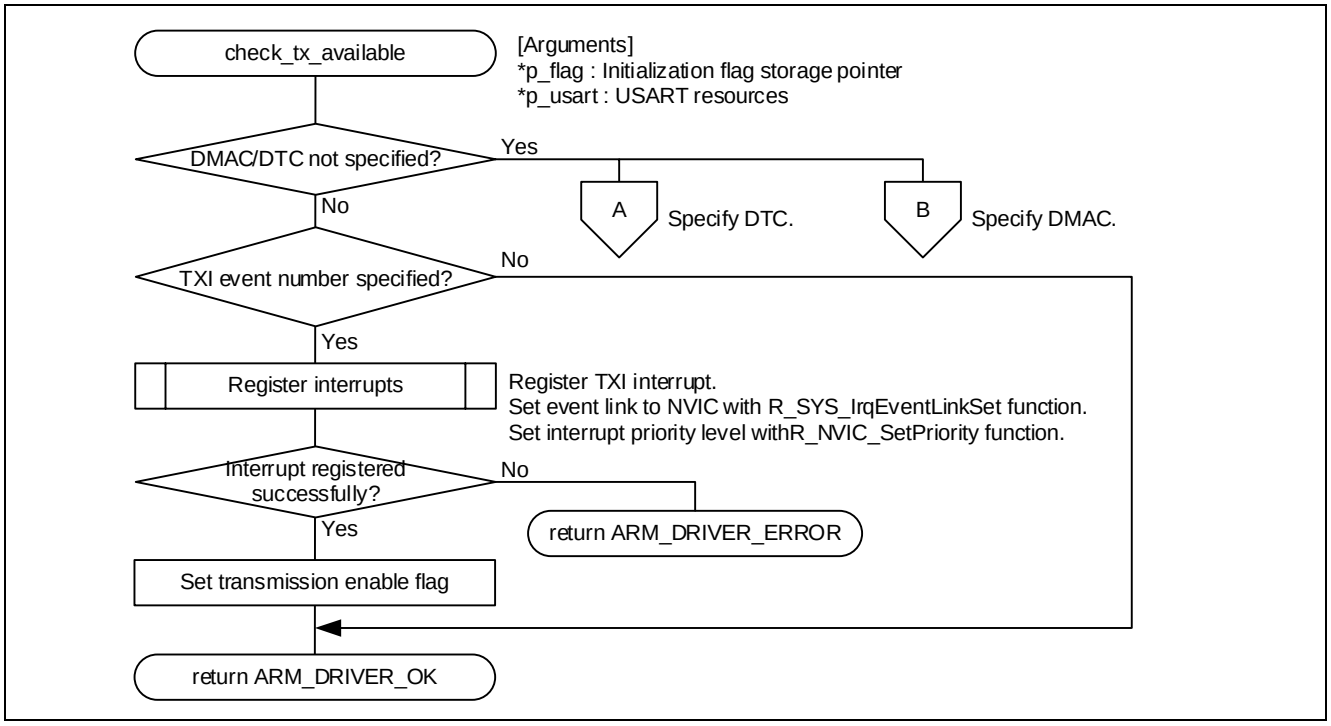
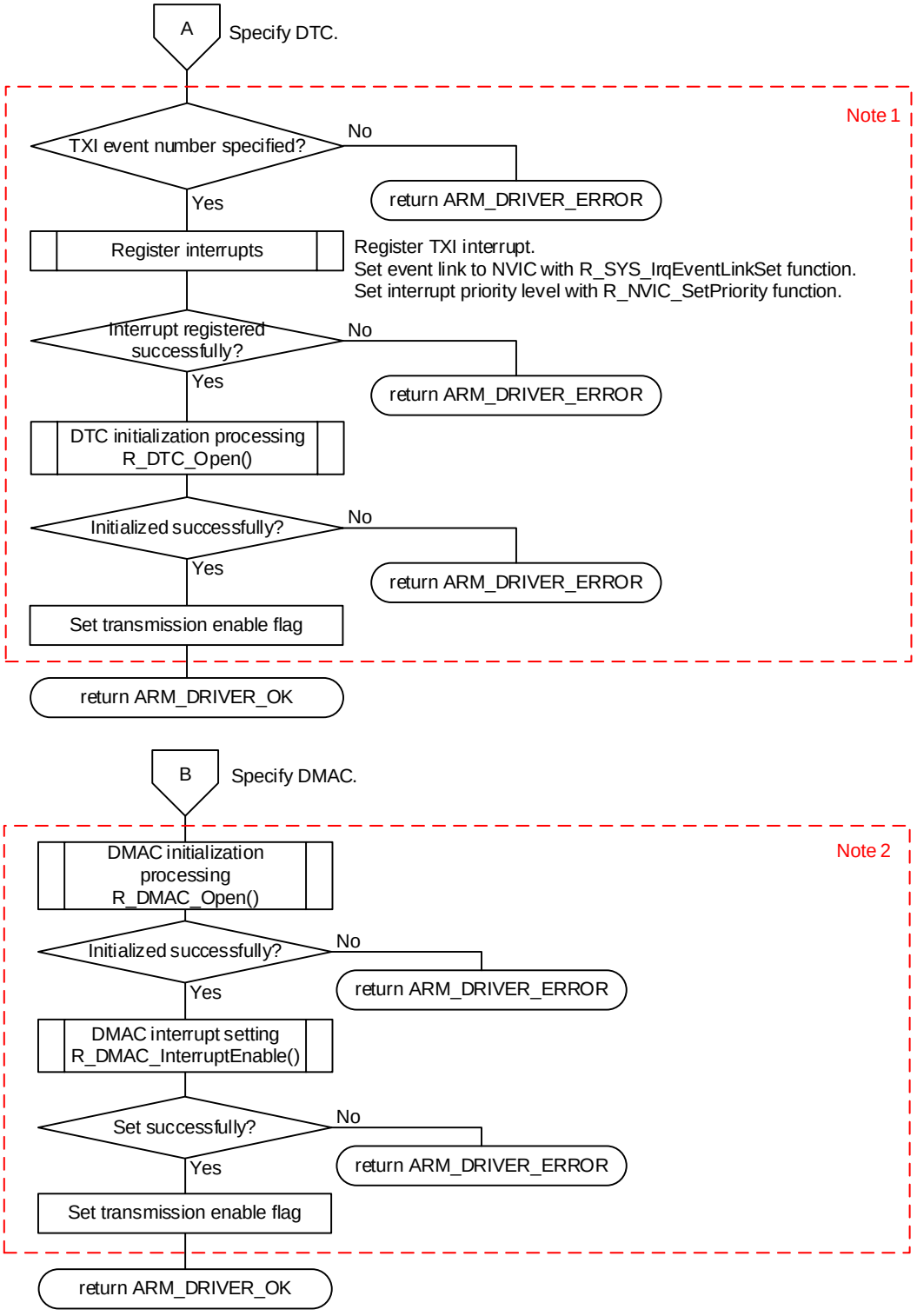


Figure 4-44 check_tx_available Function Processing Flow (1/2)



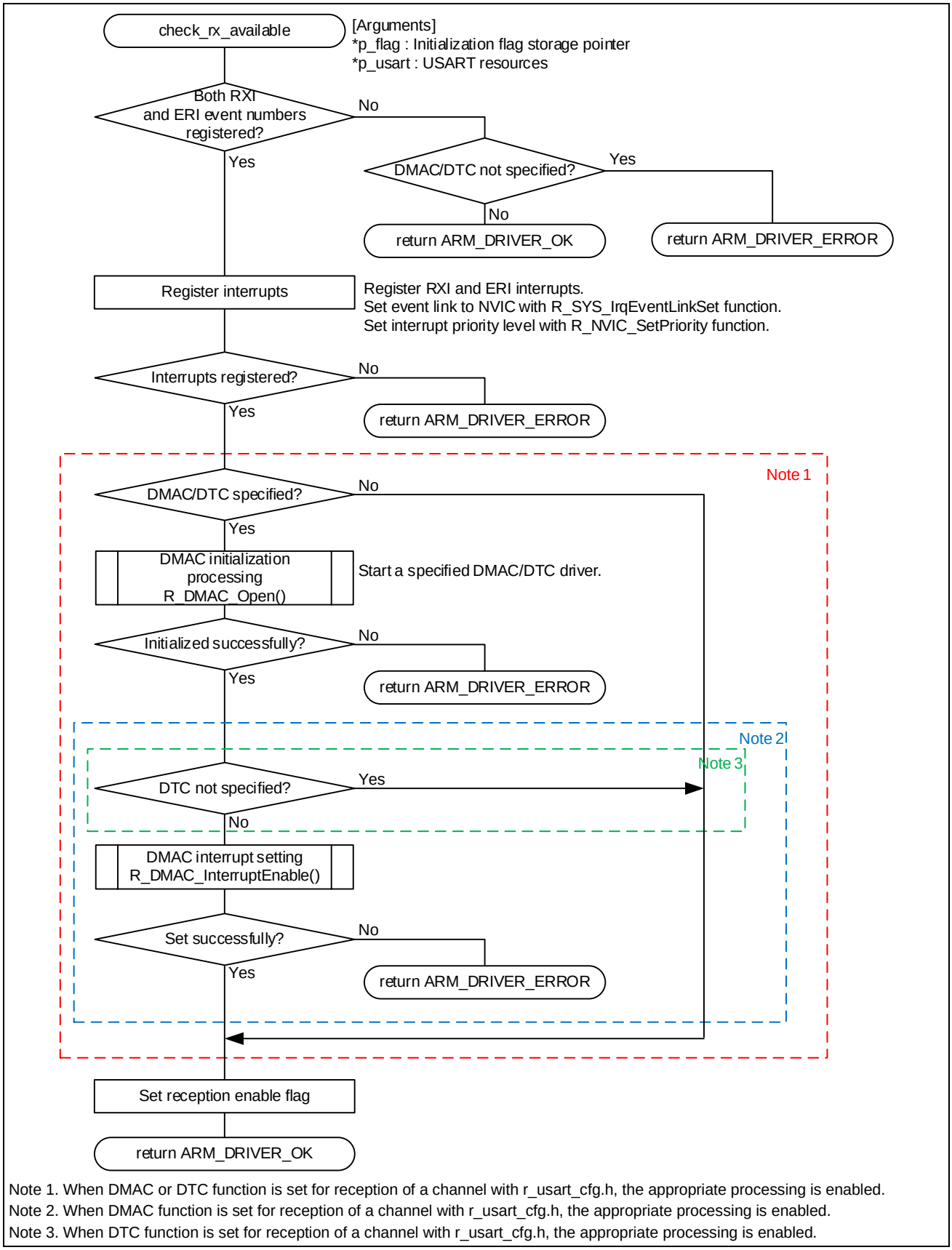
Note 1. When DTC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled.
Note 2. When DMAC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled.

Figure 4-45 check_tx_available Function Processing Flow (2/2)

4.1.27 check_rx_available Function

Table 4-29 check_rx_available Function Specifications

Format	static int32_t check_rx_available(int16_t * const p_flag, st_usart_resources_t * const p_usart)
Description	Judges whether or not reception is available.
Argument	int16_t * const p_flag: Pointer to initialization flag storage
	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.
Return value	ARM_DRIVER_OK Reception availability judgment completed
	ARM_DRIVER_ERROR Reception availability judgment failed If one of the following conditions is detected, the reception availability judgment will fail. • If RXI or ERI interrupt event link setting fails • If RXI or ERI interrupt priority level setting fails • If ERI interrupt is defined not to be used (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED) with r_system_cfg.h when DTC is used for reception. • If DMA driver initialization fails when DTC is used for reception. • If DMAC interrupt enable setting fails when DMAC is used for reception.
Remarks	



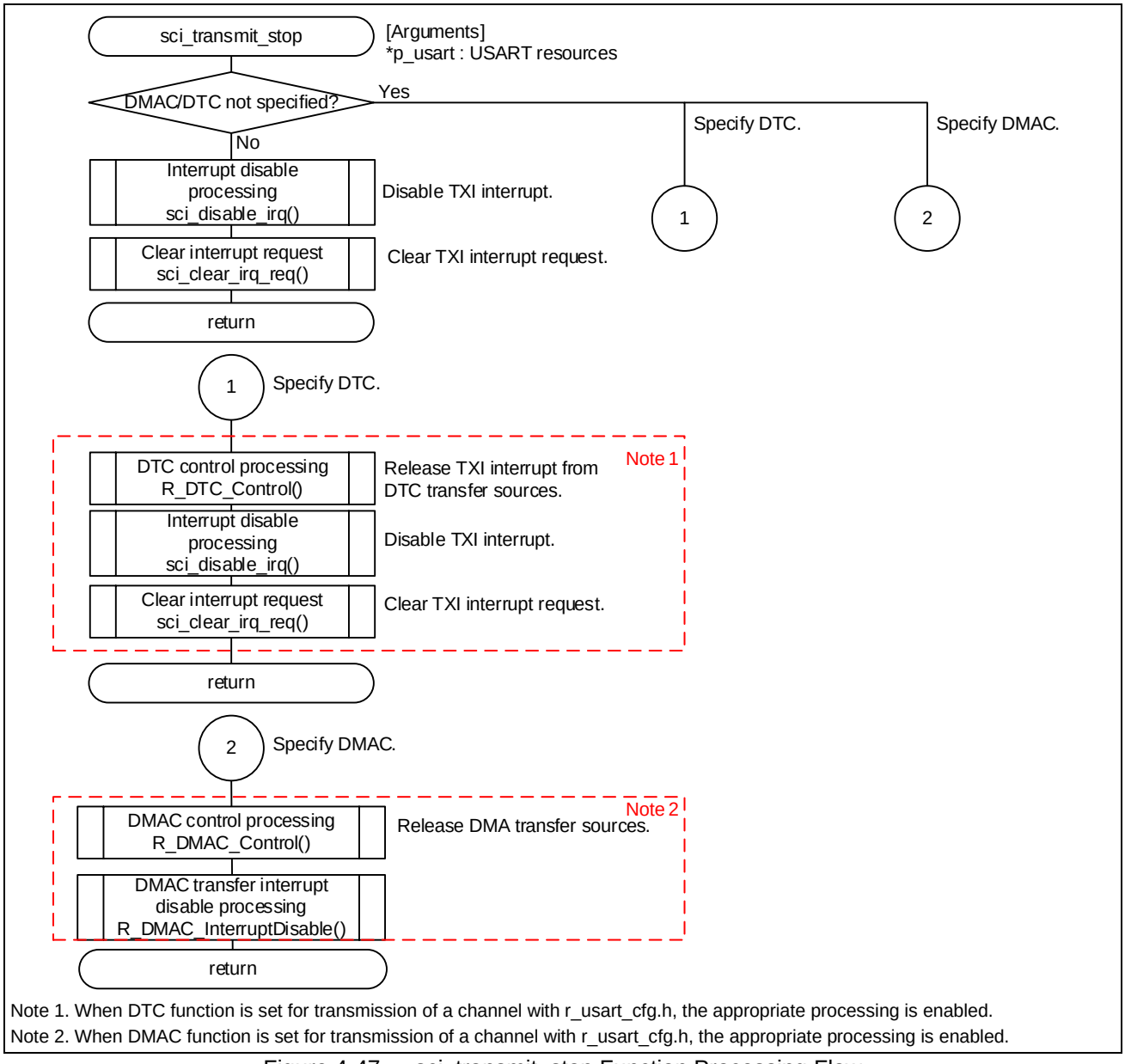
Note 1. When DMAC or DTC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled.
Note 2. When DMAC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled.
Note 3. When DTC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled.

Figure 4-46 check_rx_available Function Processing Flow

4.1.28 sci_transmit_stop Function

Table 4-30 sci_transmit_stop Function Specifications

Format	static void sci_transmit_stop(st_usart_resources_t const * const p_usart)
Description	Suspends transmission.
Argument	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	



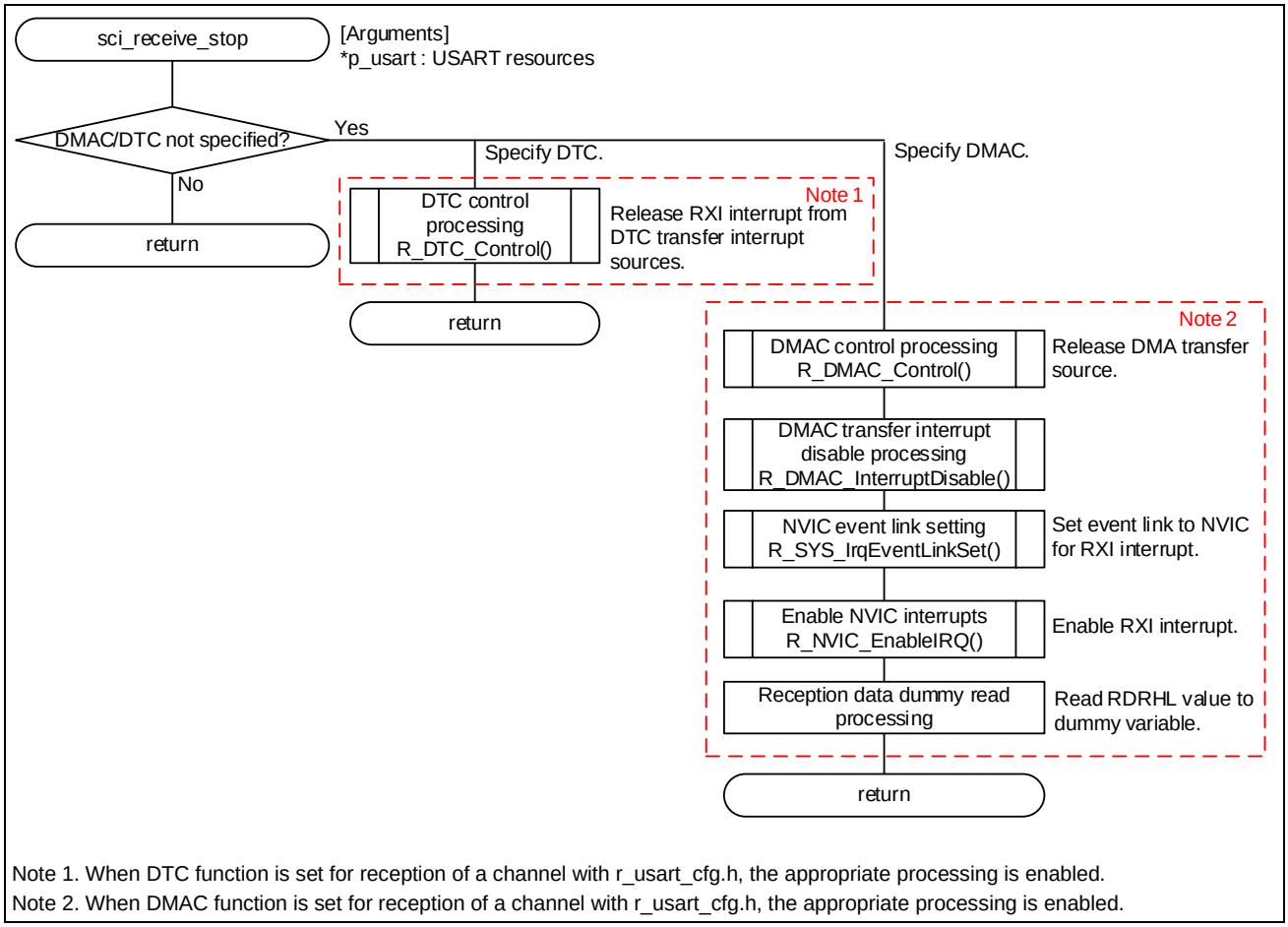
Note 1. When DTC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled.
Note 2. When DMAC function is set for transmission of a channel with r_usart_cfg.h, the appropriate processing is enabled.

Figure 4-47 sci_transmit_stop Function Processing Flow

4.1.29 sci_receive_stop Function

Table 4-31 sci_receive_stop Function Specifications

Format	static void sci_receive_stop (st_usart_resources_t const * const p_usart)
Description	Suspends reception.
Argument	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	



Note 1. When DTC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled.
Note 2. When DMAC function is set for reception of a channel with r_usart_cfg.h, the appropriate processing is enabled.

Figure 4-48 sci_receive_stop Function Processing Flow

4.1.30 dma_config_init Function

Table 4-32 dma_config_init Function Specifications

Format	static void dma_config_init(st_dma_transfer_data_cfg_t *p_cfg)
Description	Initializes the DMA driver setting structure to 0.
Argument	st_dma_transfer_data_cfg_t *p_cfg: DMA driver setting structure
Return value	None
Remarks	-

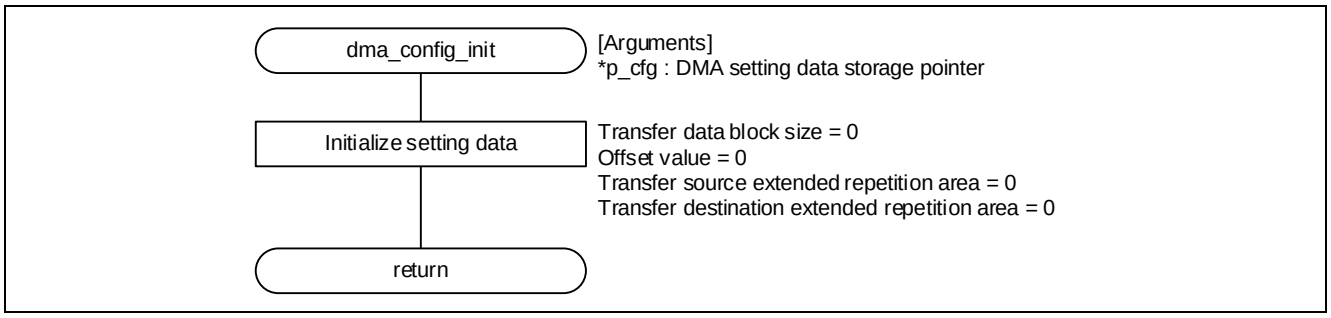


Figure 4-49 dma_config_init Function Processing Flow

4.1.31 txi_handler Function

Table 4-33 txi_handler Function Specifications

Format	static void txi_handler(st_usart_resources_t * const p_usart)
Description	TXI interrupt handling processing (when using interrupts for transmission)
Argument	st_usart_resources_t * const p_usart : Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	TXI interrupt handling processing with interrupts used for transmission

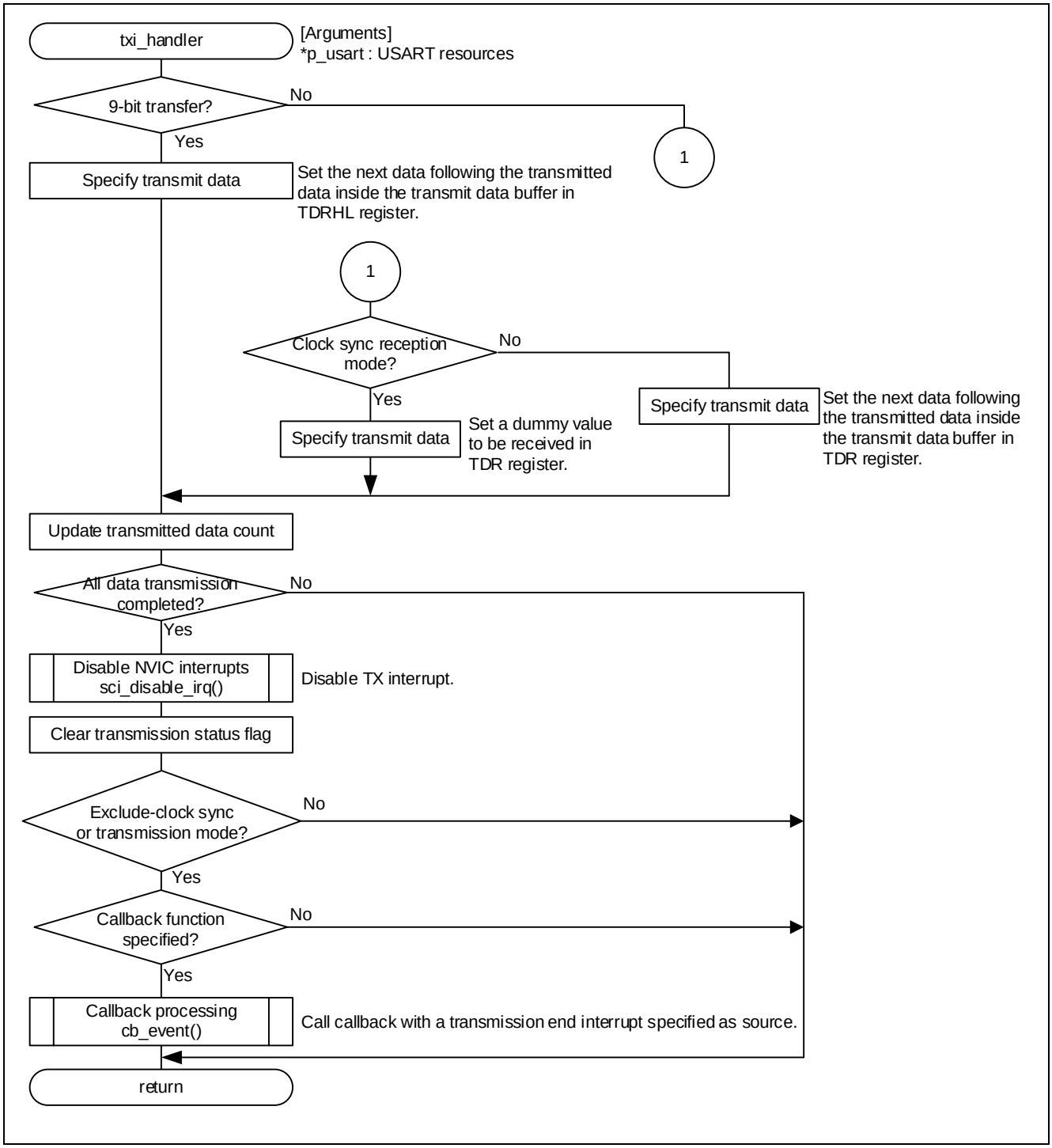


Figure 4-50 txi_handler Function Processing Flow

4.1.32 txi_dtc_handler Function

Table 4-34 txi_dtc_handler Function Specifications

Format	static void txi_dtc_handler(st_usart_resources_t * const p_usart)
Description	TXI interrupt handling processing (when using DTC for transmission)
Argument	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	TXI interrupt handling processing with DTC used for transmission

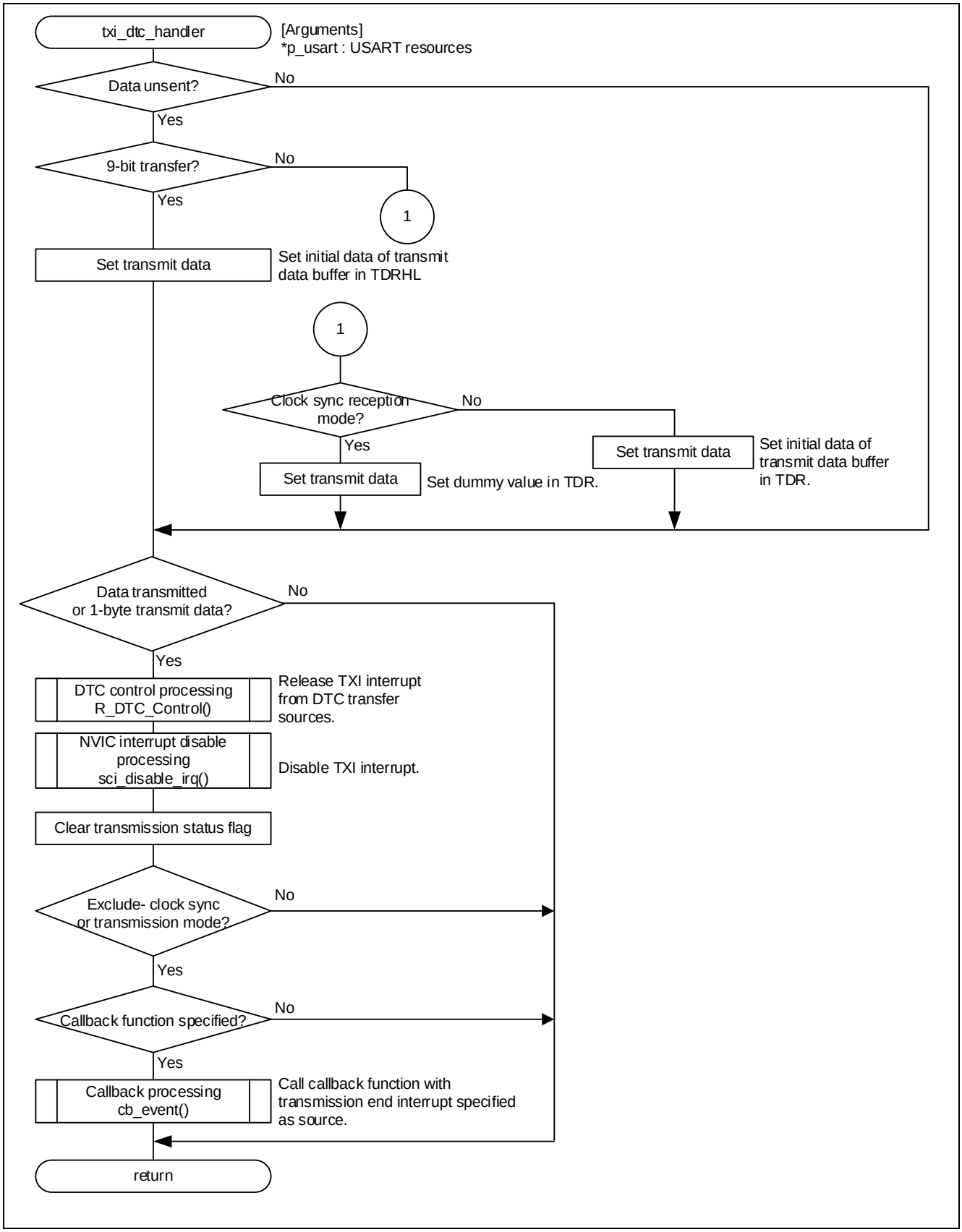


Figure 4-51 txi_dtc_handler Function Processing Flow

4.1.33 txi_dmac_handler Function

Table 4-35 txi_dmac_handler Function Specifications

Format	static void txi_dmac_handler(st_usart_resources_t * const p_usart)
Description	TXI interrupt handling processing (when using DMAC for transmission)
Argument	st_usart_resources_t * const p_usart: Resources for USART Specifies USART resources to be controlled.
Return value	None
Remarks	TXI interrupt handling processing with DMAC used for transmission

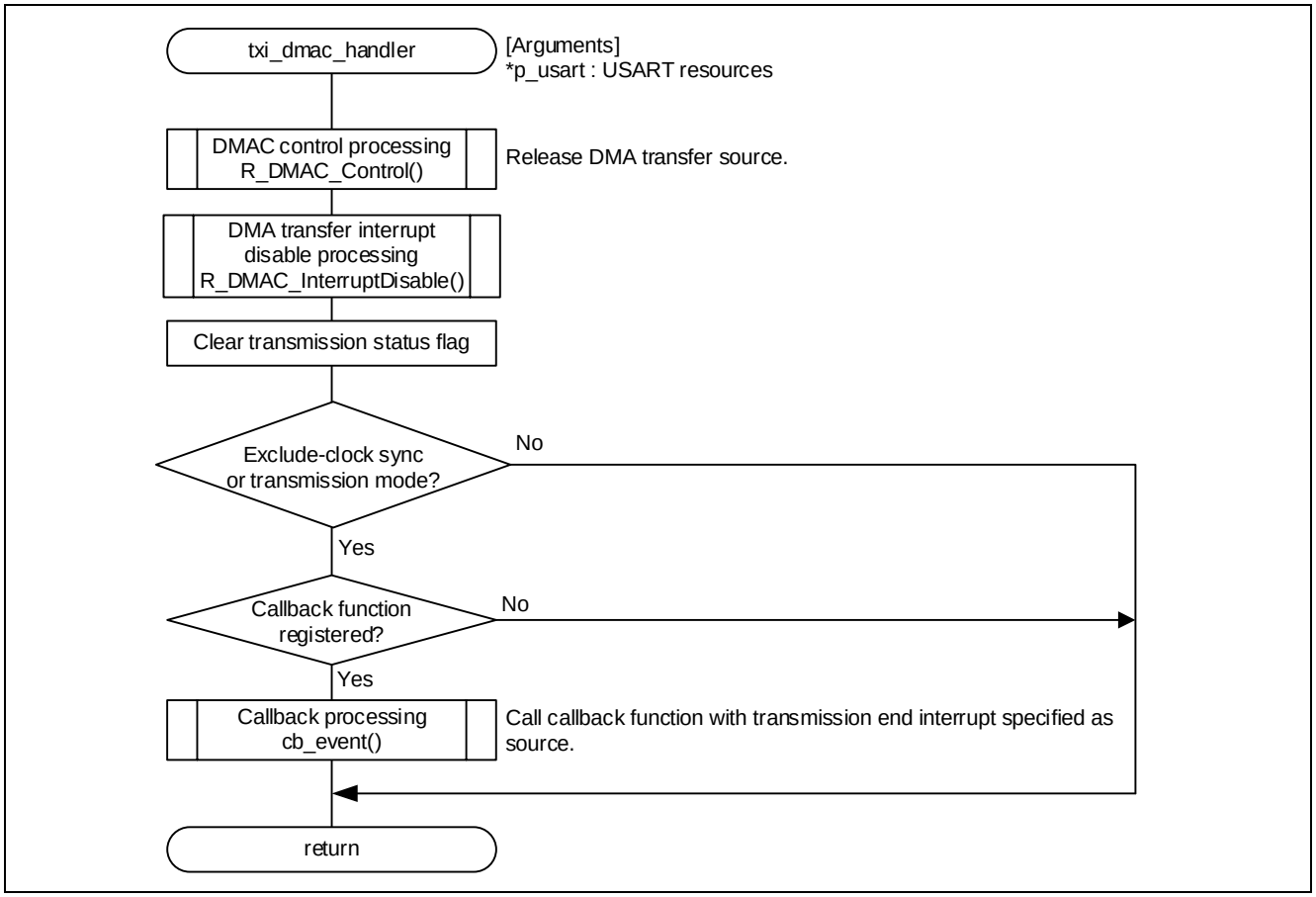


Figure 4-52 txi_dmac_handler Function Processing Flow

4.1.34 rxi_handler Function

Table 4-36 rxi_handler Function Specifications

Format	static void rxi_handler(st_usart_resources_t * const p_usart)
Description	RXI interrupt handling processing (when using interrupts for reception)
Argument	st_usart_resources_t * const p_usart : Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	RXI interrupt handling processing with interrupts used for reception

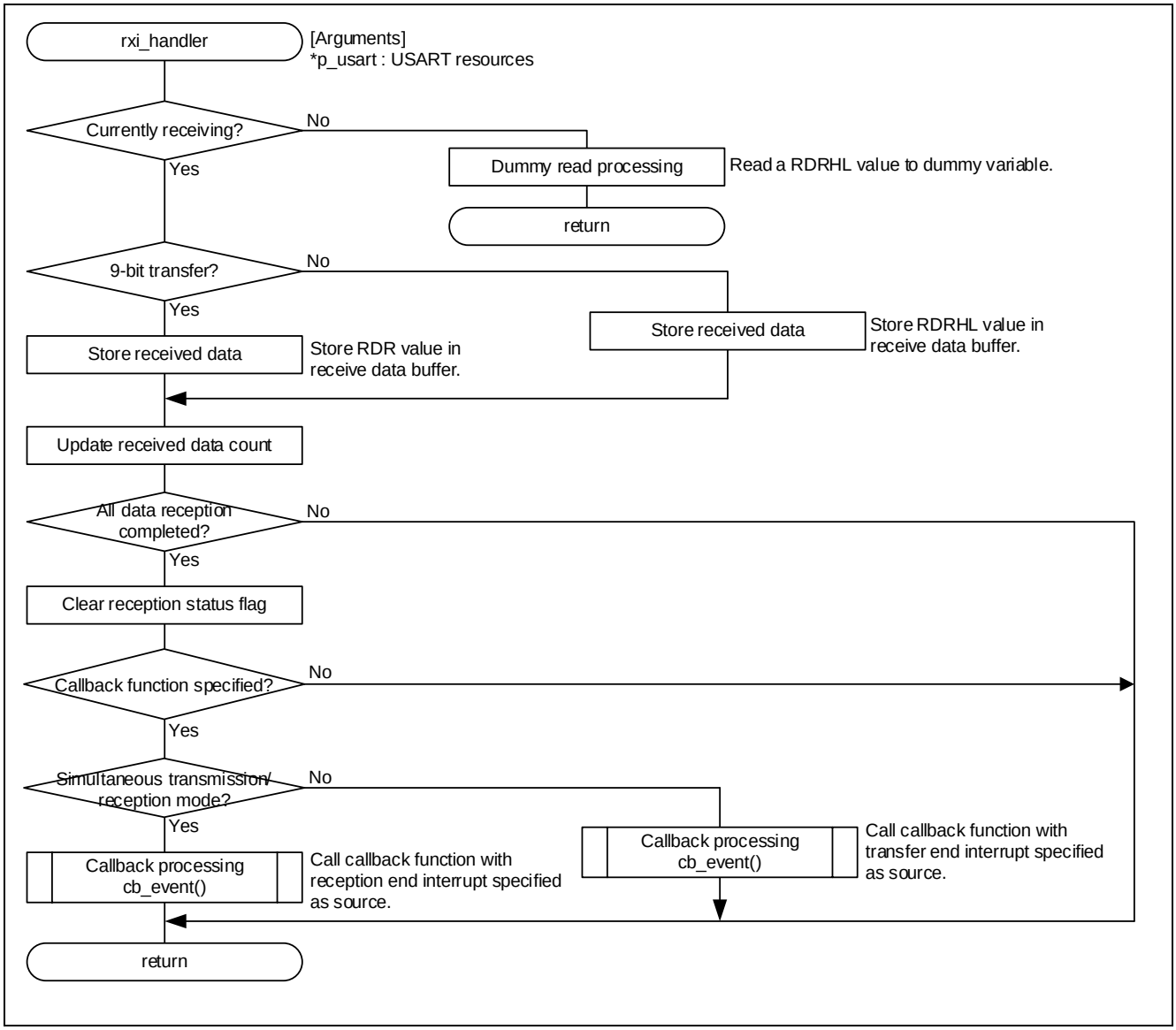


Figure 4-53 rxi_handler Function Processing Flow

4.1.35 rxi_dmac_handler Function

Table 4-37 rxi_dmac_handler Function Specifications

Format	static void rxi_dmac_handler(st_usart_resources_t * const p_usart)
Description	RXI interrupt handling processing (when using DTC or DMAC for reception)
Argument	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	RXI interrupt handling processing with DTC or DMAC used for reception

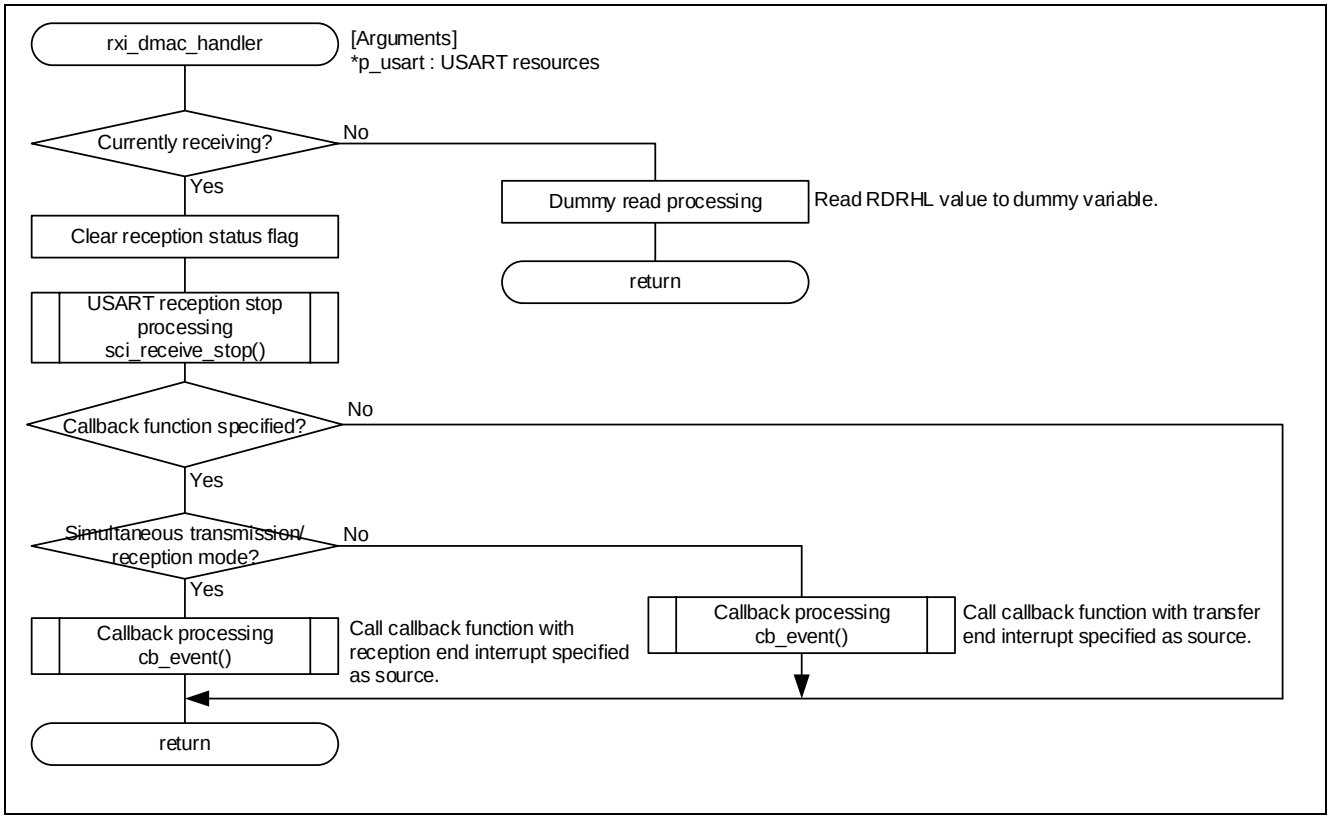


Figure 4-54 rxi_dmac_handler Function Processing Flow

4.1.36 eri_handler Function

Table 4-38 eri_handler Function Specifications

Format	static void eri_handler(st_usart_resources_t * const p_usart) // @suppress("Function length")
Description	ERI interrupt handling processing
Argument	st_usart_resources_t * const p_usart: Resources of USART Specifies USART resources to be controlled.
Return value	None
Remarks	-

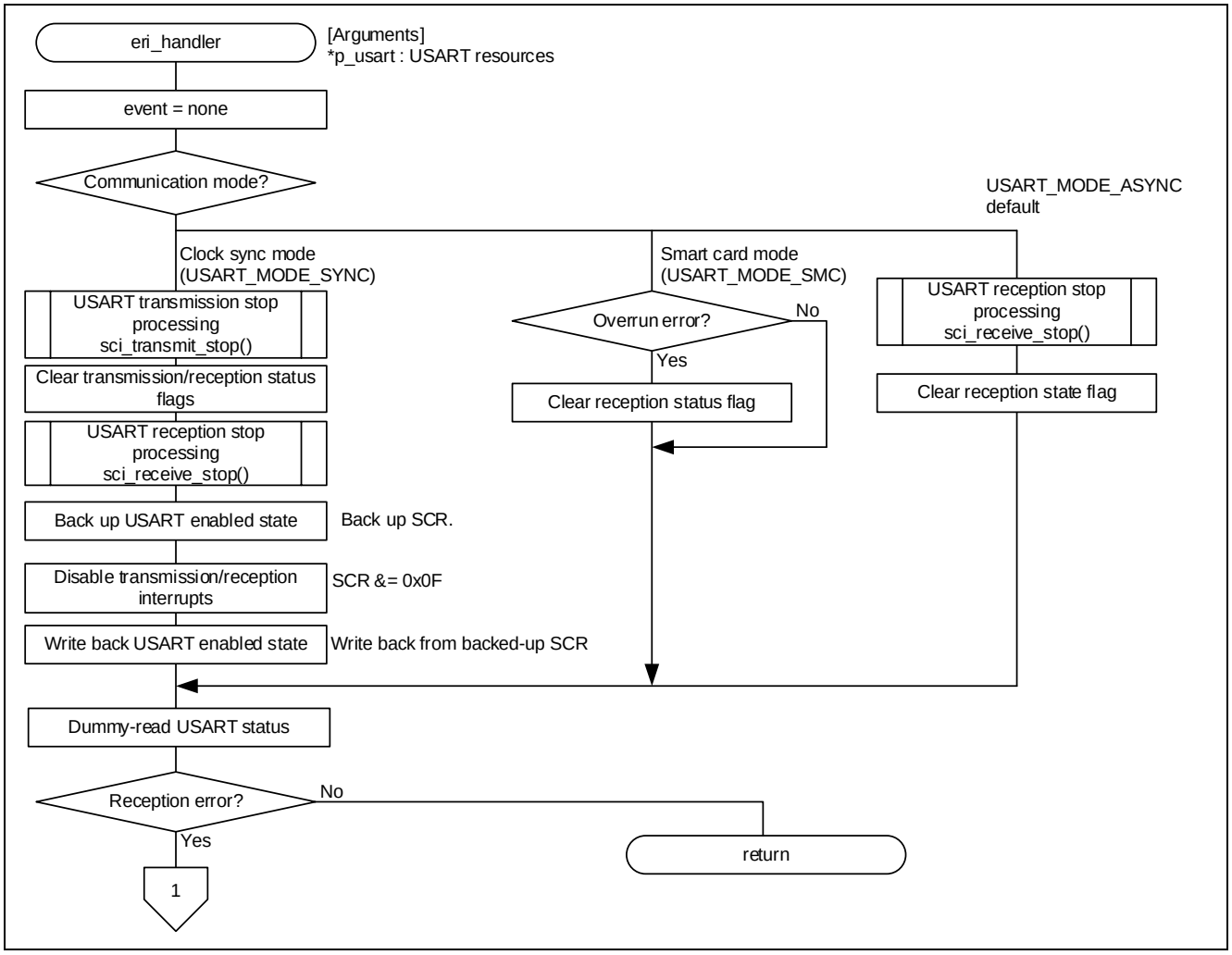


Figure 4-55 eri_handler Function Processing Flow (1/2)

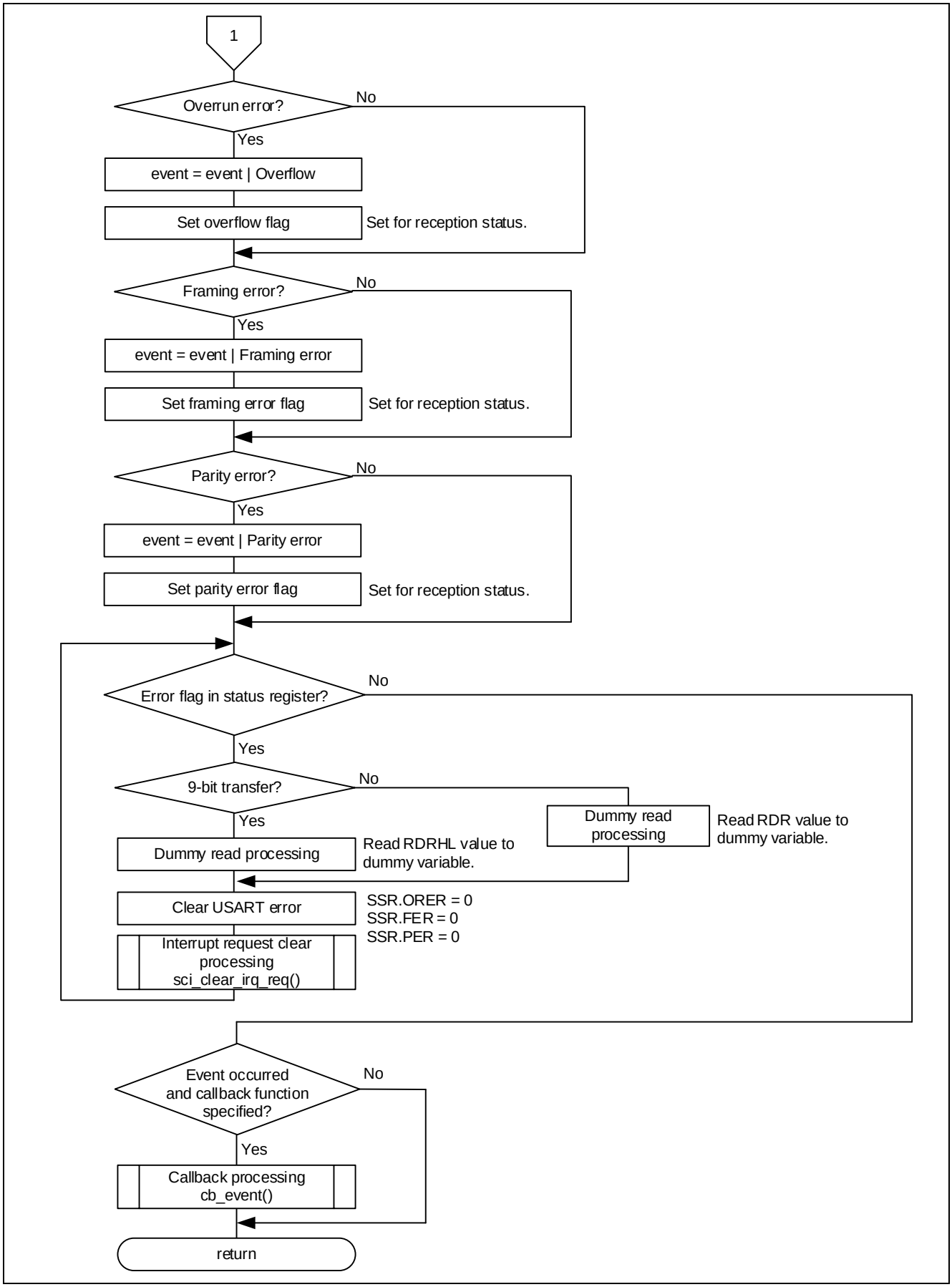


Figure 4-56 eri_handler Function Processing Flow (2/2)

4.2 Macro and Type Definitions

This section shows the definitions of the macros used in the driver and the types of them.

4.2.1 Macro Definition List

Table 4-39 List of Macro Definitions

Definition	Value	Description
R_SCI0_ENABLE	(1)	SCI0 resource enable definition
R_SCI1_ENABLE	(1)	SCI1 resource enable definition
R_SCI2_ENABLE	(1)	SCI2 resource enable definition
R_SCI3_ENABLE	(1)	SCI3 resource enable definition
R_SCI4_ENABLE	(1)	SCI4 resource enable definition
R_SCI5_ENABLE	(1)	SCI5 resource enable definition
R_SCI9_ENABLE	(1)	SCI9 resource enable definition
USART_SIZE_7	(0)	7-bit data definition
USART_SIZE_8	(1)	8-bit data definition
USART_SIZE_9	(2)	9-bit data definition
USART_SSR_ORER_MASK	(0x20U)	ORER bit mask definition
USART_SSR_FER_MASK	(0x10U)	FER bit mask definition
USART_SSR_PER_MASK	(0x08U)	PER bit mask definition
USART_RCVR_ERR_MASK	(USART_SSR_ORER_MASK | USART_SSR_FER_MASK | USART_SSR_PER_MASK)	Reception error mask definition
USART_SSR_CLR_MASK	(0xC0U)	Reception error clear mask definition
USART_FLAG_INITIALIZED	(1U << 0)	USART initialization complete flag definition
USART_FLAG_TX_AVAILABLE	(1U << 1)	Transmission enable flag definition
USART_FLAG_RX_AVAILABLE	(1U << 2)	Reception enable flag definition
USART_FLAG_POWERED	(1U << 3)	Module released flag definition
USART_FLAG_CONFIGURED	(1U << 4)	Mode setting complete flag definition
USART_FLAG_TX_ENABLED	(1U << 5)	Transmission enabled state flag definition
USART_FLAG_RX_ENABLED	(1U << 6)	Reception enabled state flag definition
USART_TXI0_IESR_VAL	(0x00000010)	IELS bit setting for TXI0
USART_RXI0_IESR_VAL	(0x00000010)	IELS bit setting for RXI0
USART_ERI0_IESR_VAL	(0x00000010)	IELS bit setting for ERI0
USART_TXI1_IESR_VAL	(0x0000001A)	IELS bit setting for TXI1
USART_RXI1_IESR_VAL	(0x0000001B)	IELS bit setting for RXI1
USART_ERI1_IESR_VAL	(0x0000001A)	IELS bit setting for ERI1
USART_TXI2_IESR_VAL	(0x0000001A)	IELS bit setting for TXI2
USART_RXI2_IESR_VAL	(0x0000001A)	IELS bit setting for RXI2
USART_ERI2_IESR_VAL	(0x00000019)	IELS bit setting for ERI2
USART_TXI3_IESR_VAL	(0x0000001C)	IELS bit setting for TXI3
USART_RXI3_IESR_VAL	(0x0000001C)	IELS bit setting for RXI3
USART_ERI3_IESR_VAL	(0x0000001B)	IELS bit setting for ERI3
USART_TXI4_IESR_VAL	(0x0000001B)	IELS bit setting for TXI4
USART_RXI4_IESR_VAL	(0x0000001B)	IELS bit setting for RXI4
USART_ERI4_IESR_VAL	(0x0000001A)	IELS bit setting for ERI4

Table 4-40 List of Macro Definitions

Definition	Value	Description
USART_TXI5_IESR_VAL	(0x0000001D)	IELS bit setting for TXI5
USART_RXI5_IESR_VAL	(0x0000001D)	IELS bit setting for RXI5
USART_ERI5_IESR_VAL	(0x0000001C)	IELS bit setting for ERI5
USART_TXI9_IESR_VAL	(0x0000001C)	IELS bit setting for TXI9
USART_RXI9_IESR_VAL	(0x0000001C)	IELS bit setting for RXI9
USART_ERI9_IESR_VAL	(0x0000001C)	IELS bit setting for ERI9
USART_TXI0_DMAC_SOURCE_ID	(0x76)	DELS bit setting for TXI0
USART_RXI0_DMAC_SOURCE_ID	(0x75)	DELS bit setting for RXI0
USART_TXI1_DMAC_SOURCE_ID	(0x7C)	DELS bit setting for TXI1
USART_RXI1_DMAC_SOURCE_ID	(0x7B)	DELS bit setting for RXI1
USART_TXI2_DMAC_SOURCE_ID	(0x81)	DELS bit setting for TXI2
USART_RXI2_DMAC_SOURCE_ID	(0x80)	DELS bit setting for RXI2
USART_TXI3_DMAC_SOURCE_ID	(0x86)	DELS bit setting for TXI3
USART_RXI3_DMAC_SOURCE_ID	(0x85)	DELS bit setting for RXI3
USART_TXI4_DMAC_SOURCE_ID	(0x8B)	DELS bit setting for TXI4
USART_RXI4_DMAC_SOURCE_ID	(0x8A)	DELS bit setting for RXI4
USART_TXI5_DMAC_SOURCE_ID	(0x90)	DELS bit setting for TXI5
USART_RXI5_DMAC_SOURCE_ID	(0x8F)	DELS bit setting for RXI5
USART_TXI9_DMAC_SOURCE_ID	(0x95)	DELS bit setting for TXI9
USART_RXI9_DMAC_SOURCE_ID	(0x94)	DELS bit setting for RXI9
REG_PRV_VALUE_CKS0	(0)	CKS0 bit position definition
REG_PRV_VALUE_CKS1	(1)	CKS1 bit position definition
REG_PRV_VALUE_BGMD	(2)	BGMD bit position definition
REG_PRV_VALUE_ABCS	(3)	ABCS bit position definition
REG_PRV_VALUE_ABCSE	(4)	ABCSE bit position definition
REG_PRV_VALUE_BCP0	(5)	BCP0 bit position definition
REG_PRV_VALUE_BCP1	(6)	BCP1 bit position definition
REG_PRV_VALUE_BCP2	(7)	BCP2 bit position definition
USART_PRV_USED_DMAC_DTC_DRV	SCI0_TRANSMIT_CONTROL | SCI0_RECEIVE_CONTROL | SCI1_TRANSMIT_CONTROL | SCI1_RECEIVE_CONTROL | SCI2_TRANSMIT_CONTROL | SCI2_RECEIVE_CONTROL | SCI3_TRANSMIT_CONTROL | SCI3_RECEIVE_CONTROL | SCI4_TRANSMIT_CONTROL | SCI4_RECEIVE_CONTROL | SCI5_TRANSMIT_CONTROL | SCI5_RECEIVE_CONTROL | SCI9_TRANSMIT_CONTROL | SCI9_RECEIVE_CONTROL	Definition for DMAC/DTC driver availability judgment
USART_PRV_USED_TX_DMAC_DTC_DRV	SCI0_TRANSMIT_CONTROL | SCI1_TRANSMIT_CONTROL | SCI2_TRANSMIT_CONTROL | SCI3_TRANSMIT_CONTROL | SCI4_TRANSMIT_CONTROL | SCI5_TRANSMIT_CONTROL | SCI9_TRANSMIT_CONTROL	Definition for DMAC/DTC transmission judgment

Table 4-41 List of Macro Definitions

Definition	Value	Description
USART_PRV_USED_RX_DMAC_DTC_DRV	SCI0_RECEIVE_CONTROL | SCI1_RECEIVE_CONTROL | SCI2_RECEIVE_CONTROL | SCI3_RECEIVE_CONTROL | SCI4_RECEIVE_CONTROL | SCI5_RECEIVE_CONTROL | SCI9_RECEIVE_CONTROL	Definition for DMAC/DTC reception judgment
USART_PRV_USED_DMAC_DRV	(USART_PRV_USED_DMAC_DTC_DRV & 0x00FF)	Definition for DMAC driver use judgment
USART_PRV_USED_TX_DMAC_DRV	(USART_PRV_USED_TX_DMAC_DTC_DRV & 0x00FF)	Definition for DMAC transmission judgment
USART_PRV_USED_RX_DMAC_DRV	(USART_PRV_USED_RX_DMAC_DTC_DRV & 0x00FF)	Definition for DMAC reception judgment
USART_PRV_USED_DTC_DRV	(USART_PRV_USED_DMAC_DTC_DRV & SCI_USED_DTC)	Definition for DTC driver use judgment
USART_PRV_USED_TX_DTC_DRV	(USART_PRV_USED_TX_DMAC_DTC_DRV & SCI_USED_DTC)	Definition for DTC transmission judgment
USART_PRV_USED_RX_DTC_DRV	(USART_PRV_USED_RX_DMAC_DTC_DRV & SCI_USED_DTC)	Definition for DTC reception judgment

4.2.2 e_usart_flow_t Definition

This definition shows a controlled flow state.

Table 4-42 List of e_usart_flow_t Definitions

Definition	Value	Description
USART_FLOW_CTS_DISABLE	0	Flow control is unused.
USART_FLOW_CTS_ENABLE	1	CTS control is used.
USART_FLOW_RTS_ENABLE	2	RTS control is used.

4.2.3 e_usart_mode_t Definition

This definition shows the operation model.

Table 4-43 List of e_usart_mode_t Definitions

Definition	Value	Description
USART_MODE_ASYNC	0	Asynchronous mode
USART_MODE_SYNC	1	Clock synchronous mode
USART_MODE_SMC	2	Smart card mode

4.2.4 e_usart_sync_t Definition

This definition shows the transmission and reception state in clock synchronous mode.

Table 4-44 List of e_usart_sync_t Definitions

Definition	Value	Description
USART_SYNC_TX_MODE	0	Operating in transmit mode
USART_SYNC_RX_MODE	1	Operating in receive mode
USART_SYNC_TX_RX_MODE	2	Operating in transmit/receive mode

4.2.5 e_usart_base_clk_t Definition

This definition shows the base clock for the relevant channel.

Table 4-45 List of e_usart_base_clk_t Definitions

Definition	Value	Description
USART_BASE_PCLKA	0	Base clock: PCLKA
USART_BASE_PCLKB	1	Base clock: PCLKB

4.3 Structure Definitions

4.3.1 st_usart_resources_t Structure

This structure configures the resources of USART.

Table 4-46 st_usart_resources_t Structure

Element Name	Type	Description
*reg	volatile SCI2_Type	Shows a target SCI register.
pin_set	r_pinset_t	Function pointer for setting pins
pin_clr	r_pinclr_t	Function pointer for releasing pins
*info	st_usart_info_t	USART status information
*cts_port	uint16_t	Software-controlled CTS pin setting (port register)
cts_pin_no	uint8_t	Software-controlled CTS pin setting (pin number)
*rts_port	uint16_t	Software-controlled RTS pin setting l (port register)
rts_pin_no	uint8_t	Software-controlled RTS pin setting (pin number)
pclk	e_usart_base_clk_t	Base clock for relevant SCI channel USART_BASE_PCLKA: PCLKA USART_BASE_PCLKB: PCLKB
lock_id	e_system_mcu_lock_t	SCI lock ID
mstp_id	e_lpm_mstp_t	SCI module stop ID
txi_irq	IRQn_Type	TXI interrupt number assigned in NVIC
rxi_irq	IRQn_Type	RXI interrupt number assigned in NVIC
eri_irq	IRQn_Type	ERI interrupt number assigned in NVIC
txi_iesr_val	uint32_t	IESR register setting for TXI interrupt
rxi_iesr_val	uint32_t	IESR register setting for RXI interrupt
eri_iesr_val	uint32_t	IESR register setting for ERI interrupt
txi_priority	uint32_t	TXI interrupt priority level
rxi_priority	uint32_t	RXI interrupt priority level
eri_priority	uint32_t	ERI interrupt priority level
*tx_dma_drv	DRIVER_DMA	DMA driver for transmission If interrupts are used for transmission, NULL is set.
tx_dma_source	uint16_t	DELS bit value set for TXI
*tx_dtc_info	st_dma_transfer_data_t	Address at which DTC transfer information for transmission is stored
*rx_dma_drv	DRIVER_DMA	DMA driver for reception If interrupts are used for reception, NULL is set.
rx_dma_source	uint16_t	DELS bit setting for RXI
*rx_dtc_info	st_dma_transfer_data_t	Address where DTC transfer information for reception is stored
txi_callback	system_int_cb_t	TXI callback function
rxi_callback	system_int_cb_t	RXI callback function
eri_callback	system_int_cb_t	ERI callback function

4.3.2 st_usart_rx_status_t Structure

This structure is used to manage the reception state of USART.

Table 4-47 st_usart_rx_status_t Structure

Element Name	Type	Description
busy	uint8_t	Reception status flag (0: Not received, 1: Reception in progress)
overflow	uint8_t	Overflow error flag (0: Overflow error not detected, 1: Overflow error detected)
framing_error	uint8_t	Framing error flag (0: Framing error not detected, 1: Framing error detected)
parity_error	uint8_t	Parity error flag (0: Parity error not detected, 1: Parity error detected)

4.3.3 st_usart_transfer_info_t Structure

This structure is used to manage the USART transmission/reception information.

Table 4-48 st_usart_transfer_info_t Structure

Element Name	Type	Description
rx_num	uint32_t	Reception size
tx_num	uint32_t	Transmission size
*rx_buf	void	Receive buffer
*tx_buf	void	Transmit buffer
rx_cnt	uint32_t	Reception count
tx_cnt	uint32_t	Transmission count
tx_def_val	uint16_t	Dummy transmit data in clock synchronous receive mode
rx_dump_val	uint8_t	Dummy read buffer
send_active	uint8_t	Transmission status flag (0: Not transmitted, 1: Transmission in progress)
sync_mode	e_usart_sync_t	Clock synchronous operation mode (enabled only in clock synchronous mode) USART_SYNC_TX_MODE: Operating in transmit mode USART_SYNC_RX_MODE: Operating in receive mode USART_SYNC_TX_RX_MODE: Operating in transmit/receive mode

4.3.4 st_usart_info_t Structure

This structure is used to manage the USART information.

Table 4-49 st_usart_info_t Structure

Element Name	Type	Description
cb_event	ARM_USART_SignalEvent_t	Callback function to be executed when an event occurs When this value is NULL, no callback function will be executed.
rx_status	st_usart_rx_status_t	USART reception state
tx_status	st_usart_transfer_info_t	USART transmission and reception information
mode	e_usart_mode_t	Operation mode USART_MODE_ASYNC: Asynchronous mode USART_MODE_SYNC: Clock synchronous mode USART_MODE_SMC: Smart card mode
data_size	uint8_t	Data size for transmission and reception USART_SIZE_7: 7-bit length USART_SIZE_8: 8-bit length USART_SIZE_9: 9-bit length
flow_mode	e_usart_flow_t	Flow control USART_FLOW_CTS_DISABLE: Flow control not used USART_FLOW_CTS_ENABLE: CTS control used USART_FLOW_RTS_ENABLE: RTS control used
flags	uint16_t	Driver status flag b0: Driver initialization state (0: Uninitialized, 1: Initialized) b1: Transmission availability (0: Transmission not available, 1: Transmission available) b2: Reception availability (0: Reception not available, 1: Reception available) b3: Module stop state (0: Module stop state, 1: Module stop released) b4: USART mode setting complete state (0: Not set, 1: Setting completed) b5: Transmission enabled state (0: Transmission prohibited, 1: Transmission enabled) b6: Reception enabled state (0: Reception prohibited, 1: Reception enabled)
baudrate	uint32_t	Baud rate setting

4.3.5 st_sci_reg_set_t Structure

This structure is used for the register setting buffers.

Table 4-50 st_sci_reg_set_t Structure

Element Name	Type	Description
smr	uint8_t	Buffer for setting SMR register
scr	uint8_t	Buffer for setting SCR register
brr	uint8_t	Buffer for setting BRR register
scmr	uint8_t	Buffer for setting SCMR register
semr	uint8_t	Buffer for setting SEMR register
spmr	uint8_t	Buffer for setting SPMR register
mddr	uint8_t	Buffer for setting MDDR register
flow_mode	e_usart_flow_t	Buffer for setting flow control
data_size	uint8_t	Buffer for setting data size

4.3.6 st_baud_divisor_t Structure

This structure is used for the baud rate calculation table.

Table 4-51 st_baud_divisor_t Structure

Element Name	Type	Description
divisor	int64_t	Frequency division ratio
reg_value	uint16_t	Register setting value

4.4 Data Table Definition

This section shows the definitions for the main data table used for USART driver processing.

4.4.1 Data Table for Calculating Baud Rate

The data table for calculating a baud rate is defined with the st_baud_divisor_t structure. The divisor element contains the frequency division ratios and the reg_value element contains the register setting values.

The reg_value element contains the values of bits set for the baud rate. Figure 4-57 shows the structure of the reg_value element.

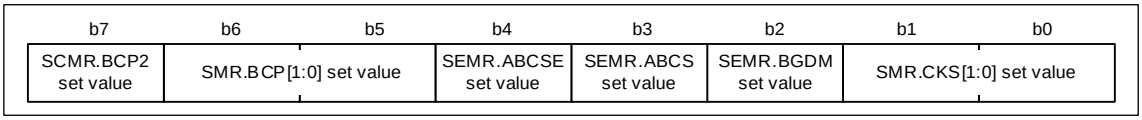


Figure 4-57 reg_value Element Structure

Baud rate calculation tables differ depending on the mode.

Table 4-52, 4-53 and 4-54 show the baud rate calculation table in asynchronous mode, in clock synchronous mode, and in smart card mode, respectively.

Table 4-52 Baud Rate Calculation Table in Asynchronous Mode (gs_async_baud)

Frequency Division Ratio	Register Setting						Description
	CKS[1:0]	BGMD	ABCS	ABCSE	BCP[1:0]	BCP2	
6	0	0	0	1	0	0	Register setting when frequency is divided by 6
8	0	1	1	0	0	0	Register setting when frequency is divided by 8
16	0	1	0	0	0	0	Register setting when frequency is divided by 16
24	1	0	0	1	0	0	Register setting when frequency is divided by 24
32	1	1	1	0	0	0	Register setting when frequency is divided by 32
64	1	1	0	0	0	0	Register setting when frequency is divided by 64
96	2	0	0	1	0	0	Register setting when frequency is divided by 96
128	2	1	1	0	0	0	Register setting when frequency is divided by 128
256	2	1	0	0	0	0	Register setting when frequency is divided by 256
384	3	0	0	1	0	0	Register setting when frequency is divided by 384
512	3	1	1	0	0	0	Register setting when frequency is divided by 512
1024	3	1	0	0	0	0	Register setting when frequency is divided by 1024
2048	3	0	0	0	0	0	Register setting when frequency is divided by 2048

Table 4-53 Baud Rate Calculation Table in Clock Synchronous Mode (gs_sync_baud)

Frequency Division Ratio	Register Setting						Description
	CKS[1:0]	BGMD	ABCS	ABCSE	BCP[1:0]	BCP2	
4	0	0	0	0	0	0	Register setting when frequency is divided by 4
16	1	0	0	0	0	0	Register setting when frequency is divided by 16
64	2	0	0	0	0	0	Register setting when frequency is divided by 64
256	3	0	0	0	0	0	Register setting when frequency is divided by 256

Table 4-54 Baud Rate Calculation Table in Smart Card Mode (gs_smc_baud)

Frequency Division Ratio	Register Setting						Description
	CKS[1:0]	BGMD	ABCS	ABCSE	BCP[1:0]	BCP2	
64	0	0	0	0	0	1	Register setting when frequency is divided by 64
128	0	0	0	0	1	1	Register setting when frequency is divided by 128
186	0	0	0	0	0	0	Register setting when frequency is divided by 186
256	1	0	0	0	0	1	Register setting when frequency is divided by 256
372	0	0	0	0	2	0	Register setting when frequency is divided by 372
512	1	0	0	0	1	1	Register setting when frequency is divided by 512
744	1	0	0	0	0	0	Register setting when frequency is divided by 744
1024	2	0	0	0	0	1	Register setting when frequency is divided by 1024
1488	1	0	0	0	2	0	Register setting when frequency is divided by 1488
2048	2	0	0	0	1	1	Register setting when frequency is divided by 2048
2976	2	0	0	0	0	0	Register setting when frequency is divided by 2976
4096	3	0	0	0	0	1	Register setting when frequency is divided by 4096
5952	2	0	0	0	2	0	Register setting when frequency is divided by 5952
8192	3	0	0	0	1	1	Register setting when frequency is divided by 8192
11904	3	0	0	0	0	0	Register setting when frequency is divided by 11904
16384	3	0	0	0	1	0	Register setting when frequency is divided by 16384
23808	3	0	0	0	2	0	Register setting when frequency is divided by 23808
32768	3	0	0	0	3	1	Register setting when frequency is divided by 32768
47616	3	0	0	0	2	1	Register setting when frequency is divided by 47616
65536	3	0	0	0	3	0	Register setting when frequency is divided by 65536

4.5 Calling External Functions

This section shows the external functions to be called from the USART driver APIs.

Table 4-55 External Functions Called from USART Driver APIs and Calling Conditions (1/3)

API	Functions Called	Conditions (Note)
Initialize	R_SYS_ResourceLock	None
	R_NVIC_GetPriority	None
	R_NVIC_SetPriority	None
	R_SYS_IrqEventLinkSet	None
	R_NVIC_ClearPendingIRQ	None
	R_SYS_IrqStatusClear	None
	R_DMAC_Open	DMAC driver was used for transmission or reception.
	R_DMAC_InterruptEnable	
	R_DMAC_Close	DMAC driver was used for transmission or reception and initialization failed.
	R_DTC_Open	DTC driver was used for transmission or reception.
	R_DTC_Close	DTC driver was used for transmission or reception and initialization failed.
Uninitialize	R_LPM_ModuleStart	The Uninitialize function was executed in module stop state.
	R_LPM_ModuleStop	None
	R_SYS_ResourceUnlock	None
	R_NVIC_ClearPendingIRQ	None
	R_SYS_IrqStatusClear	None
	R_NVIC_DisableIRQ	None
	R_SCI_Pinclr_CHn(n=0～5,9)	None
	R_DMAC_Close	DMAC driver was used for transmission or reception.
	R_DTC_Close	DTC driver was used for transmission or reception.
PowerControl	R_LPM_ModuleStart	ARM_POWER_FULL was specified (module stop state released)
	R_LPM_ModuleStop	ARM_POWER_OFF was specified (module stop state entered)
	R_NVIC_ClearPendingIRQ	
	R_SYS_IrqStatusClear	
	R_NVIC_DisableIRQ	
Send	R_NVIC_EnableIRQ	None
	R_DMAC_Create	DMAC driver was used for transmission.
	R_DMAC_InterruptEnable	
	R_DMAC_Control	
	R_DTC_Create	DTC driver was used for transmission.
	R_DTC_Control	

Note. If operation terminates due to a parameter check error, the functions may not be called even when no condition is specified.

Table 4-56 External Functions Called from USART Driver APIs and Calling Conditions (2/3)

API	Functions Called	Conditions (Note)
Receive	R_NVIC_EnableIRQ	None
	R_DMAC_Create	DMAC driver was used for reception, or DMAC driver was used for transmission in clock synchronous mode (with transmission enabled).
	R_DMAC_InterruptEnable	
	R_DMAC_Control	
	R_DMAC_InterruptDisable	DMAC driver was used for transmission in clock synchronous mode (with transmission enabled) and DMAC setting failed in transmission process.
	R_SYS_IrqEventLinkSet	DMAC driver was used for transmission or reception.
	R_NVIC_ClearPendingIRQ	
	R_SYS_IrqStatusClear	
	R_NVIC_DisableIRQ	
	R_DTC_Create	DTC driver was used for reception, or DTC driver was used for transmission in clock synchronous mode (with transmission enabled).
	R_DTC_Control	
Transfer	R_NVIC_EnableIRQ	None
	R_DMAC_Create	DMAC driver was used for transmission or reception.
	R_DMAC_InterruptEnable	
	R_DMAC_Control	
	R_SYS_IrqEventLinkSet	
	R_NVIC_ClearPendingIRQ	
	R_SYS_IrqStatusClear	
	R_NVIC_DisableIRQ	
	R_DMAC_InterruptDisable	DMAC driver was used for transmission and DMAC setting failed in transmission process.
	R_DTC_Create	DTC driver was used for reception, or DTC driver was used in clock synchronous mode (with transmission enabled).
	R_DTC_Control	
GetTxCount	R_DMAC_GetTransferByte	When the DMAC driver is used for transmission processing and the transmission size is not 1 byte
	R_DTC_GetTransferByte	When a DTC driver is used for transmission processing
GetRxCount	R_DMAC_GetTransferByte	When a DMAC driver is used for reception processing
	R_DTC_GetTransferByte	When a DTC driver is used for reception processing

Note. If operation ends due to a parameter check error, the functions will not be called in some cases even when there is no condition for executing them.

Table 4-57 External Functions Called from USART Driver APIs and Calling Conditions (3/3)

API	Functions Called	Conditions
Control	R_SYS_PeripheralClockFreqGet R_SYS_SystemClockFreqGet	Any of the following commands was executed: • ARM_USART_MODE_ASYNCHRONOUS • ARM_USART_MODE_SYNCHRONOUS_MASTER • ARM_USART_MODE_SMART_CARD
	R_NVIC_ClearPendingIRQ R_SYS_IrqStatusClear	Any of the following commands was executed: • ARM_USART_ABORT_SEND • ARM_USART_ABORT_RECEIVE • ARM_USART_ABORT_TRANSFER
	R_DMAC_Control R_DMAC_InterruptDisable	DMAC driver was used for transmission or reception and any of the following commands was executed: • ARM_USART_ABORT_SEND • ARM_USART_ABORT_RECEIVE • ARM_USART_ABORT_TRANSFER
	R_SYS_IrqEventLinkSet R_NVIC_EnableIRQ	DMAC driver was used for reception and any of the following commands was executed: • ARM_USART_ABORT_RECEIVE • ARM_USART_ABORT_TRANSFER
	R_NVIC_ClearPendingIRQ R_SYS_IrqStatusClear	Reception was disabled with either of the following commands: • ARM_USART_ABORT_RECEIVE • ARM_USART_ABORT_TRANSFER
	R_SCI_Pinset_CHn(n=0～5,9) R_NVIC_EnableIRQ	Transmission or reception was enabled with any of the following commands: • ARM_USART_CONTROL_TX • ARM_USART_CONTROL_RX • ARM_USART_CONTROL_TX_RX
	R_SCI_Pinclr_CHn(n=0～5,9)	Transmission and reception were disabled with any of the following commands: • ARM_USART_CONTROL_TX • ARM_USART_CONTROL_RX • ARM_USART_CONTROL_TX_RX
	R_NVIC_DisableIRQ	Transmission or reception was disabled with any of the following commands: • ARM_USART_CONTROL_TX • ARM_USART_CONTROL_RX • ARM_USART_CONTROL_TX_RX
GetStatus	-	-
SetModemControl	-	-
GetModemStatus	-	-
GetVersion	-	-
GetCapabilities	-	-

5. Usage Notes

5.1 Arguments

Initialize all the elements of the structures used for the arguments of each function to 0 before use.

5.2 Registering USART Interrupts to NVIC

It is necessary to register the interrupts used for communication control to NVIC in r_system_cfg.h.

For details, see section 2.4 Communication Control and NVIC Interrupt Setting, Communication Control.

5.3 Power supply open control register (VOCR) setting

Use this driver after setting the power supply open control register (VOCR).

The VOCR register prevents indefinite inputs from entering the power domain that is not supplied with power. For this reason, the VOCR register is set to shut off the input signal after reset. In this state, the input signal is not propagated inside the device. For details, refer to “VOCR (Power Supply Open Control) Register Settings” in “RE01 1500KB, 256KB Group Startup Guide to Development Using CMSIS Package R01AN4660”.

5.4 Coding for Reception in Clock Synchronous and Smart Card Modes

To enable transmission and reception simultaneously in clock synchronous mode, enable TE and RE using the procedure shown below. TE and RE need to be enabled simultaneously because of hardware restrictions. If the shown procedure is not used, the only the one that is set first will be effective.

To enable reception only, also use the same procedure to write dummy data. Figure 5-1 shows a coding example for reception in clock synchronous mode.

```
#include "R_Driver_USART.h"

static void usart_callback(uint32_t event);
// USART driver instance ( SCI0 )
extern ARM_DRIVER_USART Driver_USART0;
static ARM_DRIVER_USART *gsp_sci0_dev = &Driver_USART0;
// Receive data
static uint8_t rx_data[6];

main()
{
    uint32_t arg;
    /* Clock synchronous master mode */
    arg = ARM_USART_MODE_SYNCHRONOUS_MASTER |
          ARM_USART_CPOL0 | ARM_USART_CPHA0 |
          ARM_USART_FLOW_CONTROL_NONE,;

    (void)gsp_sci0_dev->Initialize(usart_callback);             /* Initialize USART driver */
    (void)gsp_sci0_dev->PowerControl(ARM_POWER_FULL);    /* Release USART module stop */
    (void)gsp_sci0_dev->Control(arg, 100000);               /* Clock synchronous master mode
(100kbps) */
    (void)gsp_sci0_dev->Control(ARM_USART_CONTROL_TX_RX,1);  /* Enables transmission and
  reception at the same time */

    (void) gsp_sci0_dev->Receive(rx_data, 6);  /* Start reception */
    while(1);
}

/******************************************************************************************
* callback function
******************************************************************************************/
static void usart_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_USART_EVENT_SEND_COMPLETE:
        {
         /* Describe the processing to be performed when transmission terminates normally */
        }
        break;

        case ARM_USART_EVENT_RECEIVE_COMPLETE:
        {
            /* Describe the processing to be performed when reception terminates normally */
        }
        break;

        case ARM_USART_EVENT_TRANSFER_COMPLETE:
        {
            /* Describe the processing to be performed when transmission/reception
               terminates normally */
        }
        break;

        default:
        {
            /* Describe the processing to be performed when a communication error occurs */
        }
        break;
    }
}
```

Figure 5-1 Coding Example for Reception in Clock Synchronous Mode

5.5 Pin Configuration

The pins to be used by this driver are set and released respectively with the R_SCI_Pinset_CHn (n=0 to 5, 9) and R_SCI_Pinclr_CHn functions in pin.c. The R_SCI_Pinset_CHn function is called when transmission or reception is enabled by the Control function. The R_SCI_Pinclr_CHn function is called when transmission and reception are disabled by the Control function, PowerControl function, or Uninitialize function.

Select the pin to be used by editing the the R_SCI_Pinset_CHn and R_SCI_Pinclr_CHn functions (n = 0 to 5 and 9) of pin.c.

Figure 5-2 to Figure 5-4 show the coding examples for setting the pin configuration in which SCI0 is used in asynchronous mode.

```
/*****************************************************************************//**
* @brief This function sets Pin of SCI0.
*****************************************************************************/
/* Function Name : R_SCI_Pinset_CH0 */
void R_SCI_Pinset_CH0(void)  // @suppress("API function naming") @suppress("Function length")
{
    /* Disable protection for PFS function (Set to PWPR register) */
    R_SYS_RegisterProtectDisable(SYSTEM_REG_PROTECT_MPC);

//    /* CTS0 : P107 */
//    PFS->P107PFS_b.PMR  = 0U;
//    PFS->P107PFS_b.ASEL = 0U;
//    PFS->P107PFS_b.ISEL = 0U;
//    PFS->P107PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P107PFS_b.PMR  = 1U;

    /* CTS0 : P500 */
//    PFS->P500PFS_b.ASEL = 0U;
//    PFS->P500PFS_b.ISEL = 0U;
//    PFS->P500PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P500PFS_b.PMR  = 1U;

    /* CTS0 : P704 */
//    PFS->P704PFS_b.ASEL = 0U;
//    PFS->P704PFS_b.ISEL = 0U;
//    PFS->P704PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P704PFS_b.PMR  = 1U;

////    /* TXD0 : P106 */
//    PFS->P106PFS_b.PMR  = 0U;
//    PFS->P106PFS_b.ASEL = 0U;
//    PFS->P106PFS_b.ISEL = 0U;

//    /* When using SCI in I2C mode, set the pin to NMOS Open drain. */
////    PFS->P106PFS_b.NCODR = 1U;
////    PFS->P106PFS_b.PCODR = 0U;
//    PFS->P106PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P106PFS_b.PMR  = 1U;

    /* TXD0 : P013 */
//    PFS->P013PFS_b.ASEL = 0U;
//    PFS->P013PFS_b.ISEL = 0U;

    /* When using SCI in I2C mode, set the pin to NMOS Open drain. */
////    PFS->P013PFS_b.NCODR = 1U;
////    PFS->P013PFS_b.PCODR = 0U;
//    PFS->P013PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P013PFS_b.PMR  = 1U;

    /* Set P703 as TXD0 */
    /* TXD0 : P703 */
    PFS->P703PFS_b.ASEL = 0U;
    PFS->P703PFS_b.ISEL = 0U;

    /* When using SCI in I2C mode, set the pin to NMOS Open drain. */
//    PFS->P703PFS_b.NCODR = 1U;
//    PFS->P703PFS_b.PCODR = 0U;
    PFS->P703PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
    PFS->P703PFS_b.PMR  = 1U;
```

Figure 5-2 Coding Example for Setting Pin Configuration (1/3)

```
//    /* RXD0 : P105 */
//    PFS->P105PFS_b.PMR  = 0U;
//    PFS->P105PFS_b.ASEL = 0U;
//    PFS->P105PFS_b.ISEL = 0U;

//    /* When using SCI in I2C mode, set the pin to NMOS Open drain. */
////    PFS->P105PFS_b.NCODR = 1U;
////    PFS->P105PFS_b.PCODR = 0U;
//    PFS->P105PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P105PFS_b.PMR  = 1U;

    /* RXD0 : P014 */
//    PFS->P014PFS_b.ASEL = 0U;
//    PFS->P014PFS_b.ISEL = 0U;

    /* When using SCI in I2C mode, set the pin to NMOS Open drain. */
////    PFS->P014PFS_b.NCODR = 1U;
////    PFS->P014PFS_b.PCODR = 0U;
//    PFS->P014PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P014PFS_b.PMR  = 1U;

    /* Set P702 as RXD0 */
    /* RXD0 : P702 */
    PFS->P702PFS_b.ASEL = 0U;
    PFS->P702PFS_b.ISEL = 0U;

    /* When using SCI in I2C mode, set the pin to NMOS Open drain. */
//    PFS->P702PFS_b.NCODR = 1U;
//    PFS->P702PFS_b.PCODR = 0U;
    PFS->P702PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
    PFS->P702PFS_b.PMR  = 1U;

//    /* SCK0 : P104 */
//    PFS->P104PFS_b.PMR  = 0U;
//    PFS->P104PFS_b.ASEL = 0U;
//    PFS->P104PFS_b.ISEL = 0U;
//    PFS->P104PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P104PFS_b.PMR  = 1U;

    /* SCK0 : P015 */
//    PFS->P015PFS_b.ASEL = 0U;
//    PFS->P015PFS_b.ISEL = 0U;
//    PFS->P015PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P015PFS_b.PMR  = 1U;

    /* SCK0 : P700 */
//    PFS->P700PFS_b.ASEL = 0U;
//    PFS->P700PFS_b.ISEL = 0U;
//    PFS->P700PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P700PFS_b.PMR  = 1U;

    /* Enable protection for PFS function (Set to PWPR register) */
    R_SYS_RegisterProtectEnable(SYSTEM_REG_PROTECT_MPC);

}/* End of function R_SCI_Pinset_CH0() */
```

Figure 5-3 Coding Example for Setting Pin Configuration (2/3)

```
/*****************************************************************************//**
* @brief This function clears the pin setting of SCI0.
*****************************************************************************/
/* Function Name : R_SCI_Pinclr_CH0 */
void R_SCI_Pinclr_CH0(void)  // @suppress("API function naming")
{
    /* Disable protection for PFS function (Set to PWPR register) */
    R_SYS_RegisterProtectDisable(SYSTEM_REG_PROTECT_MPC);

//    /* SCK0 : P104 */
//    PFS->P104PFS &= R_PIN_PRV_CLR_MASK;

    /* SCK0 : P015 */
//    PFS->P015PFS &= R_PIN_PRV_CLR_MASK;

    /* SCK0 : P700 */
//    PFS->P700PFS &= R_PIN_PRV_CLR_MASK;

/* P702 を RXD0 に設定 */
//    /* RXD0 : P105 */
//    PFS->P105PFS &= R_PIN_PRV_CLR_MASK;

    /* RXD0 : P014 */
//    PFS->P014PFS &= R_PIN_PRV_CLR_MASK;

/* Release RXD0 pin */
    /* RXD0 : P702 */
    PFS->P702PFS &= R_PIN_PRV_CLR_MASK;

//    /* TXD0 : P106 */
//    PFS->P106PFS &= R_PIN_PRV_CLR_MASK;

    /* TXD0 : P013 */
//    PFS->P013PFS &= R_PIN_PRV_CLR_MASK;

/* Release TXD0 pin */
    /* TXD0 : P703 */
    PFS->P703PFS &= R_PIN_PRV_CLR_MASK;

//    /* CTS0 : P107 */
//    PFS->P107PFS &= R_PIN_PRV_CLR_MASK;

    /* CTS0 : P500 */
//    PFS->P500PFS &= R_PIN_PRV_CLR_MASK;

    /* CTS0 : P704 */
//    PFS->P704PFS &= R_PIN_PRV_CLR_MASK;

    /* Enable protection for PFS function (Set to PWPR register) */
    R_SYS_RegisterProtectEnable(SYSTEM_REG_PROTECT_MPC);
}/* End of function R_SCI_Pinclr_CH0() */
```

Figure 5-4 Coding Example for Setting Pin Configuration (3/3)

5.6 Notes on using DMAC control for transmission control

When DMAC is used for transmission control, the first byte of transmission data is written in the Send function. Therefore, when only 1 byte transmission is performed, the callback function is called in the Send function.

If 1 byte is sent again with the callback function, it becomes a recursive function (calls the callback function within the callback function).

5.7 RTS Control using SetModemControl Function

When controlling software RTS by using the SetModemControl function, set the pins to be used as RTS by using USARTn_RTS_PORT and USARTn_RTS_PIN (n = 0 to 5 and 9) of the r_usart_cfg.h file. In addition, for the flow control used to select the operating mode by the Control function, set ARM_USART_FLOW_CONTROL_NONE, (CTS/RTS not used) or ARM_USART_FLOW_CONTROL_CTS (CTS function used); do not set hardware RTS control.

Figure 5-5 shows an example for setting the software RTS control pins to PORT508 with USART0.

```
/* When using SetModemControl function, please define USART_RTS_PORT and USART_RTS_PIN. */
#define USART0_RTS_PORT       (PORT5->PODR)      ///< Used RTS0 port
#define USART0_RTS_PIN         8                   ///< Used RTS0 pin number
//#define USART1_RTS_PORT      (PORT0->PODR)      ///< Used RTS1 port
#define USART1_RTS_PIN         0                   ///< Used RTS1 pin number
//#define USART2_RTS_PORT      (PORT0->PODR)      ///< Used RTS2 port
#define USART2_RTS_PIN         0                   ///< Used RTS2 pin number
//#define USART3_RTS_PORT      (PORT0->PODR)      ///< Used RTS3 port
#define USART3_RTS_PIN         0                   ///< Used RTS3 pin number
//#define USART4_RTS_PORT      (PORT0->PODR)      ///< Used RTS4 port
#define USART4_RTS_PIN         0                   ///< Used RTS4 pin number
//#define USART5_RTS_PORT      (PORT0->PODR)      ///< Used RTS5 port
#define USART5_RTS_PIN         0                   ///< Used RTS5 pin number
//#define USART9_RTS_PORT      (PORT0->PODR)      ///< Used RTS9 port
#define USART9_RTS_PIN         0                   ///< Used RTS9 pin number
```

Figure 5-5 Example for Setting Software RTS Control Pins

5.8 CTS Pin Status Acquisition using GetModemStatus Function

When acquiring the CTS pin status by using the GetModemStatus function, set the pins to be used as CTS by using USARTn_CTS_PORT and USARTn_CTS_PIN (n = 0 to 5 and 9) of the r_usart_cfg.h file. In addition, for the flow control used to select the operating mode by the Control function, set ARM_USART_FLOW_CONTROL_NONE, (CTS/RTS not used) or ARM_USART_FLOW_CONTROL_RTS (CTS function used); do not set hardware RTS control.

Figure 5-6 shows an example for setting the CTS pins to be used in the GetModemStatus function to PORT509 with USART0.

```
/* When using GetModemStatus function, please define USART_CTS_PORT and USART_CTS_PIN. */
#define USART0_CTS_PORT       (PORT5->PIDR)      ///< Used CTS0 port
#define USART0_CTS_PIN         9                  ///< Used CTS0 pin number
//#define USART1_CTS_PORT      (PORT0->PIDR)      ///< Used CTS1 port
#define USART1_CTS_PIN         0                  ///< Used CTS1 pin number
//#define USART2_CTS_PORT      (PORT0->PIDR)      ///< Used CTS2 port
#define USART2_CTS_PIN         0                  ///< Used CTS2 pin number
//#define USART3_CTS_PORT      (PORT0->PIDR)      ///< Used CTS3 port
#define USART3_CTS_PIN         0                  ///< Used CTS3 pin number
//#define USART4_CTS_PORT      (PORT0->PIDR)      ///< Used CTS4 port
#define USART4_CTS_PIN         0                  ///< Used CTS4 pin number
//#define USART5_CTS_PORT      (PORT0->PIDR)      ///< Used CTS5 port
#define USART5_CTS_PIN         0                  ///< Used CTS5 pin number
//#define USART9_CTS_PORT      (PORT0->PIDR)      ///< Used CTS9 port
#define USART9_CTS_PIN         0                  ///< Used CTS9 pin number
```

Figure 5-6 Example for Setting CTS Pins to Be Used in GetModemStatus Function

6. Reference Documents

User's Manual: Hardware

RE01 1500KB Group User's Manual: Hardware R01UH0796
RE01 256KB Group User's Manual: Hardware R01UH0894
(The latest version can be downloaded from the Renesas Electronics website.)

RE01 Group CMSIS Package Startup Guide

RE01 1500KB, 256KB Group Startup Guide to Development Using CMSIS Package R01AN4660

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

Technical Update/Technical News

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

User's Manual: Development Tools

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

Revision History

Rev.	Date	Description	
		Page	**Summary**
1.00	Oct.10.19	—	First edition issued
1.01	Dec.02.2019	15, 166~169	Modification to comment out default pin setting of pin.c
1.03	Feb.18.2020	— 128, 132,133 31,32, 39~41, 47~49, 52~54, 60~62, 65~68 Program	Clerical error correction Modified the flow of the terminal setting procedure according to the program Corrected erroneous description of transmission and transmission timing diagrams Following two fixes Fixed the problem that output becomes HI-z for several cycles when setting pins. - Revised the driver setting procedure inside the driver - Fixed the problem that the reception complete interrupt does not occur when DTC is selected for USART CH9 reception control. - Corrected the judgment formula in internal function sci9_rxi_interrupt ()
1.04	Mar.5.2020	— Program 256KB	Compatible with 256KB group Modified to match 256KB IO definition - Changed access method to TDR register - Changed access method to RDR register - Changed access method to TDRHL register - Changed access method to RDRHL register
1.05	Apr.17.2020	Program	Changed the configuration so that it can be built without DMAC and DTC drivers.

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

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

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

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

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

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

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(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.4.0-1 November 2017)

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/.