RE01 1500KB、256KB グループ

CMSIS ドライバ USART 仕様書

要旨

本書では、RE01 1500KB、256KB グループ CMSIS software package の USART ドライバ（以下、USART ドライバ）の詳細仕様を説明します。

動作確認デバイス

RE01 1500KB グループ

RE01 256KB グループ

目次

1. 概要

USART ドライバは、Arm 社の基本ソフトウェア規定 CMSIS に準拠した RE01 1500KB および 256KB グループ用のドライバです。本ドライバでは以下の周辺機能を使用します。

表 1-1 R_USART ドライバで使用する周辺機能

周辺機能	内容
シリアルコミュニケーション インタフェース(SCI)	SCI を使用し、調歩同期式および同期式シリアル通信を実現します
データトランスファ コントローラ(DTC)(注)	DTC 制御選択時、送信データレジスタ(TDR)へのデータ書き込みや受信データレジスタ(RDR)からのデータ読み取りに DTC を使用します
DMA コントローラ(DMAC)(注)	DMAC 制御選択時、送信データレジスタ(TDR)へのデータ書き込みや受信データレジスタ(RDR)からのデータ読み取りに DMAC を使用します

注　通信制御に DMAC もしくは DTC を指定した場合のみ使用します。詳細は「2.4 通信制御および NVIC 割り込み設定」を参照してください。

2. ドライバ構成

本章では、本ドライバ使用するために必要な情報を記載します。

2.1 ファイル構成

USART ドライバは CMSIS Driver Package の CMSIS_Driver に該当し、CMSIS ファイル格納ディレクトリ内の”Driver_USART.h”と、ベンダ独自ファイル格納ディレクトリ内の ”r_usart_cmsis_api.c”、”r_usart_cmsis_api.h”、”r_usart_cfg.h”、”R_Driver_USART.h” 、”pin.c”、”pin.h”の 7 個のファイルで構成されます。各ファイルの役割を表 2-1 に、ファイル構成を図 2-1 に示します。

表 2-1 R_USART ドライバ 各ファイルの役割

ファイル名	内容
Driver_USART.h	CMSIS Driver 標準ヘッダファイルです
R_Driver_USART.h	CMSIS Driver の拡張ヘッダファイルです USART ドライバを使用する場合は、本ファイルをインクルードする必要があります
r_usart_cmsis_api.c	ドライバソースファイルです ドライバ関数の実体を用意します USART ドライバを使用する場合は、本ファイルをビルドする必要があります
r_usart_cmsis_api.h	ドライバヘッダファイルです ドライバ内で使用するマクロ／型／プロトタイプ宣言が定義されています
r_usart_cfg.h	コンフィグレーション定義ファイルです ユーザが設定可能なコンフィグレーション定義を用意します
pin.c	端子設定ファイルです 各種機能の端子割り当て処理を用意します
pin.h	端子設定ヘッダファイルです

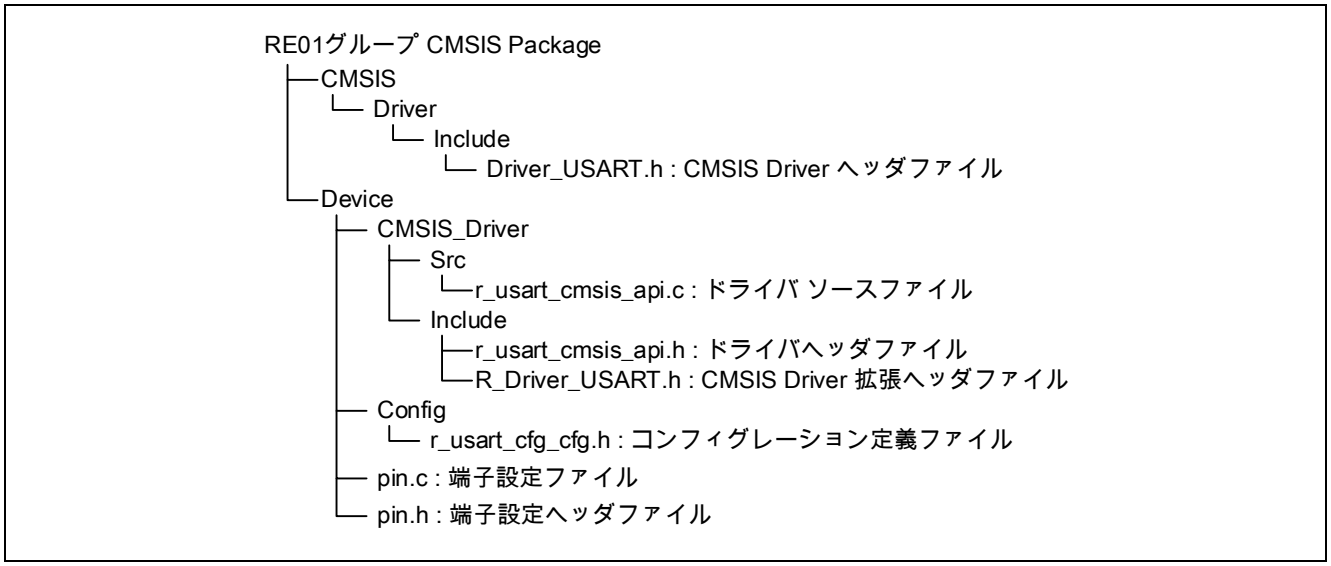


図 2-1 USART ドライバファイル構成

2.2 ドライバ API

USART ドライバはチャネル別にインスタンスを用意しています。ドライバを使用する場合は、インスタンス内の関数ポインタを使用して API にアクセスしてください。USART ドライバのインスタンス一覧を表 2-2 に、インスタンスの宣言例を図 2-2 に、インスタンスに含まれる API を表 2-3 に、USART ドライバへのアクセス例を図 2-3～図 2-6 に示します。

表 2-2 USART ドライバのインスタンス一覧

インスタンス	内容
ARM_DRIVER_USART Driver_USART0	SCI0 を使用する場合のインスタンス
ARM_DRIVER_USART Driver_USART1	SCI1 を使用する場合のインスタンス
ARM_DRIVER_USART Driver_USART2	SCI2 を使用する場合のインスタンス
ARM_DRIVER_USART Driver_USART3	SCI3 を使用する場合のインスタンス
ARM_DRIVER_USART Driver_USART4	SCI4 を使用する場合のインスタンス
ARM_DRIVER_USART Driver_USART5	SCI5 を使用する場合のインスタンス
ARM_DRIVER_USART Driver_USART9	SCI9 を使用する場合のインスタンス

```
#include "R_Driver_USART.h"

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

図 2-2 USART ドライバ インスタンス宣言例

表 2-3 USART ドライバ API

API	内容	参照
Initialize	USART ドライバの初期化（RAM の初期化、NVIC への割り込み登録）を行います また、送受信に DMA を使用する場合は DMA の初期化も実施します	4.1.3
Uninitialize	USART ドライバを解放（端子の解放）します モジュールストップ状態でない場合、モジュールストップ状態への遷移を、送受信に DMA 使用時、DMA ドライバの解放も行います	4.1.4
PowerControl	USART のモジュールストップ状態の解除または遷移を行います	4.1.5
Send	送信を開始します	4.1.6
Receive	受信を開始します	4.1.7
Transfer	送受信を開始します	4.1.8
GetTxCount	送信数を取得します	4.1.9
GetRxCount	受信数を取得します	4.1.10
Control	USART の制御コマンドを実行します 制御コマンドについては「表 2-4 制御コマンド一覧」を参照	4.1.11
GetStatus	USART の状態を取得します	4.1.12
SetModemControl	モデム制御を行います	4.1.13
GetModemStatus	モデム状態を取得します	4.1.14
GetVersion	USART ドライバのバージョンを取得します	4.1.1
GetCapabilities	USART ドライバの機能を取得します	4.1.2

表 2-4 制御コマンド一覧

コマンド	内容
ARM_USART_MODE_ASYNCHRONOUS	調歩同期モードにて USART を初期化します データビット長定義、パリティ機能定義、ストップビット長定義、フロー制御定義と組み合わせて指定してください 第 2 引数にはボーレートを指定してください
ARM_USART_MODE_SYNCHRONOUS_MASTER	クロック同期マスタモードにて USART を初期化します フロー制御定義、クロック極性定義、クロック位相定義と組み合わせて指定してください 第 2 引数にはボーレートを指定してください
ARM_USART_MODE_SYNCHRONOUS_SLAVE	クロック同期スレーブモードにて USART を初期化します フロー制御定義、クロック極性定義、クロック位相定義と組み合わせて指定してください
ARM_USART_MODE_SMART_CARD	スマートカードモードにて USART を初期化します パリティ機能定義と組み合わせて指定してください 第 2 引数にはボーレートを指定してください
ARM_USART_SET_DEFAULT_TX_VALUE	クロック同期モードの受信動作時に出力する送信データ（デフォルトデータ）を設定します 第 2 引数にはデフォルトデータの値を設定してください
ARM_USART_SET_SMART_CARD_CLOCK	スマートカードクロックの出力有無を設定します 第 2 引数に現在のボーレートを設定するとクロック出力が許可されます。0 を設定すると、クロック出力を停止します
ARM_USART_CONTROL_SMART_CARD_NACK	スマートカードモードの NACK 出力を許可にします 第 2 引数には”1”(許可)のみ有効です
ARM_USART_CONTROL_TX	送信を許可、または禁止にします(注 2、注 3) 第 2 引数には”1”(許可)または”0”(禁止)を設定してください
ARM_USART_CONTROL_RX	受信を許可、または禁止にします(注 2、注 3) 第 2 引数には”1”(許可)または”0”(禁止)を設定してください
ARM_USART_CONTROL_TX_RX	送受信を許可、または禁止にします(注 2、注 3) 第 2 引数には”1”(許可)または”0”(禁止)を設定してください
ARM_USART_ABORT_SEND	送信を中断します
ARM_USART_ABORT_RECEIVE	受信を中断します
ARM_USART_ABORT_TRANSFER	送受信を中断します
ARM_USART_MODE_SINGLE_WIRE	使用禁止(注 1)
ARM_USART_MODE_IRDA	使用禁止(注 1)
ARM_USART_SET_IRDA_PULSE	使用禁止(注 1)
ARM_USART_SET_SMART_CARD_GUARD_TIME	使用禁止(注 1)
ARM_USART_CONTROL_BREAK	使用禁止(注 1)

注1. USART ドライバではサポートしていません。Control 関数で本定義を指定した場合、ARM_DRIVER_ERROR_UNSUPPORTED を返します。

注2. クロック同期で送信・受信を同時に行う場合は、ARM_USART_CONTROL_TX_RX を使用してください。H/W の制限で TE、RE を同時に許可する必要があります。ARM_USART_CONTROL_TX、ARM_USART_CONTROL_RX を個別に設定した場合、先に設定した側のみ設定されます。

注3. 送信禁止、受信禁止状態から許可設定した場合、USART で使用する端子の設定も行います。
また、送信許可状態、または受信許可状態から送受信禁止状態となった場合、USART で使用する端子の解除も行います。

調歩同期モード、クロック同期モード、スマートカードモードで USART を初期化するときに使用する機能は、Control 関数でモード設定時に組み合わせて指定します。

使用例）

調歩同期モードをビット長 8、パリティビットなし、1 ストップビット、フロー制御なし、9600bps で初期化する場合

```
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);
```

USART で指定できる機能を表 2-5～表 2-10 に、各モードでの USART アクセス例を図 2-3～図 2-6 に示します。機能を指定しなかった場合は、(デフォルト)と記載されている機能が有効になります。

表 2-5 データビット長定義一覧（調歩同期モードで有効）

コマンド	内容
ARM_USART_DATA_BITS_7	データビット長 7 ビット
ARM_USART_DATA_BITS_8(デフォルト)	データビット長 8 ビット
ARM_USART_DATA_BITS_9	データビット長 9 ビット
ARM_USART_DATA_BITS_5	使用禁止(注)
ARM_USART_DATA_BITS_6	使用禁止(注)

注　USART ドライバではサポートしていません。

表 2-6 パリティ機能定義一覧（調歩同期モード、スマートカードモードで有効）

コマンド	内容
ARM_USART_PARITY_NONE,(デフォルト)	パリティビットなし(注)
ARM_USART_PARITY_EVEN	偶数パリティ
ARM_USART_PARITY_ODD	奇数パリティ

注　スマートカードモードで ARM_USART_PARITY_NONE,は使用できません。本機能を指定した場合は ARM_USART_ERROR_PARITY を返します。

表 2-7 ストップビット長定義一覧（調歩同期モードで有効）

コマンド	内容
ARM_USART_STOP_BITS_1 (デフォルト)	1 ストップビット
ARM_USART_STOP_BITS_2	2 ストップビット
ARM_USART_STOP_BITS_1_5	使用禁止(注)
ARM_USART_STOP_BITS_0_5	使用禁止(注)

注　USART ドライバではサポートしていません。

表 2-8 フロー制御定義一覧（調歩同期モード、クロック同期モードで有効）

コマンド	内容
ARM_USART_FLOW_CONTROL_NONE, (デフォルト)	フロー制御なし
ARM_USART_FLOW_CONTROL_RTS	RTS フロー制御
ARM_USART_FLOW_CONTROL_CTS	CTS フロー制御
ARM_USART_FLOW_CONTROL_CTS_RTS	使用禁止(注)

注　USART ドライバではサポートしていません。

表 2-9 クロック極性定義一覧（クロック同期モードで有効）

コマンド	内容
ARM_USART_CPOL0 (デフォルト)	クロック極性反転なし
ARM_USART_CPOL1	クロック極性反転あり

表 2-10 クロック位相遅れ定義一覧（クロック同期モードで有効）

コマンド	内容
ARM_USART_CPHA0 (デフォルト)	クロック遅延なし
ARM_USART_CPHA1	クロック遅延あり

```
#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);              /* USART ドライバ初期化 */
    (void)sci0Drv->PowerControl(ARM_POWER_FULL);            /* USART のモジュールストップ解除 */
    (void)sci0Drv->Control(ARM_USART_MODE_ASYNCHRONOUS |    /* 調歩同期モード */
                           ARM_USART_DATA_BITS_8       |    /* データビット長 8 ビット */
                           ARM_USART_PARITY_NONE,      |    /* パリティ機能なし */
                           ARM_USART_STOP_BITS_2       |    /* 2 ストップビット */
                           ARM_USART_FLOW_CONTROL_NONE,,    /* フロー制御なし */
                           9600);                           /* 通信速度: 9600bps */
    (void)sci0Drv->Control(ARM_USART_CONTROL_TX_RX,1);      /* 送受信許可 */

    (void)sci0Drv->Receive(&rx_data[0],3);                  /* 3 バイト受信開始 */
    (void)sci0Drv->Send(&tx_data[0],3);                     /* 3 バイト送信開始 */

    while(1);
}

/**********************************************************************************************
* callback function
**********************************************************************************************/
static void usart_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_USART_EVENT_SEND_COMPLETE:
        {
            /* 正常に送信完了した場合の処理を記述 */
        }
        break;

        case ARM_USART_EVENT_RECEIVE_COMPLETE:
        {
            /* 正常に受信完了した場合の処理を記述 */
        }
        break;

        default:
        {
            /* 通信異常が発生した場合の処理を記述 */
        }
        break;
    }

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

図 2-3 USART ドライバへのアクセス例（調歩同期モード）

```
#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);              /* USART ドライバ初期化 */
    (void)sci0Drv->PowerControl(ARM_POWER_FULL);            /* USART のモジュールストップ解除 */
    (void)sci0Drv->Control(ARM_USART_MODE_SYNCHRONOUS_MASTER  |  /* クロック同期マスタ */
                        ARM_USART_CPOL0 |                   /* クロック反転なし */
                        ARM_USART_CPHA0 |                   /* クロック位相遅れなし */
                        ARM_USART_FLOW_CONTROL_NONE,;,      /* フロー制御なし */
                        100000);                            /* 通信速度: 100kbps */
    (void)sci0Drv->Control(ARM_USART_CONTROL_TX_RX,1);      /* 送受信許可 */

    (void)sci0Drv->Transfer (&tx_data[0], &rx_data[0], 3); /* 3 バイト送受信開始 */

    while(1);
}

/******************************************************************************************
* callback function
******************************************************************************************/
static void usart_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_USART_EVENT_SEND_COMPLETE:
        {
            /* 正常に送信完了した場合の処理を記述 */
        }
        break;

        case ARM_USART_EVENT_RECEIVE_COMPLETE:
        {
            /* 正常に受信完了した場合の処理を記述 */
        }
        break;

        case ARM_USART_EVENT_TRANSFER_COMPLETE:
        {
            /* 正常に送受信完了した場合の処理を記述 */
        }
        break;

        default:
        {
            /* 通信異常が発生した場合の処理を記述 */
        }
        break;
    }

}   /* End of function usart callback() */
```

図 2-4 USART ドライバへのアクセス例（クロック同期マスタモード）

```
#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);              /* USART ドライバ初期化 */
    (void)sci0Drv->PowerControl(ARM_POWER_FULL);            /* USART のモジュールストップ解除 */
    (void)sci0Drv->Control(ARM_USART_MODE_SYNCHRONOUS_SLAVE  |  /* クロック同期スレーブ */
                        ARM_USART_CPOL0 |                   /* クロック反転なし */
                        ARM_USART_CPHA0 |                   /* クロック位相遅れなし */
                        ARM_USART_FLOW_CONTROL_NONE,;,      /* フロー制御なし */
                        0);
    (void)sci0Drv->Control(ARM_USART_CONTROL_TX_RX,1);      /* 送受信許可 */

    (void)sci0Drv->Transfer (&tx_data[0], &rx_data[0], 3); /* 3 バイト送受信開始 */

    while(1);
}

/*****************************************************************************************
* callback function
*****************************************************************************************/
static void usart_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_USART_EVENT_SEND_COMPLETE:
        {
            /* 正常に送信完了した場合の処理を記述 */
        }
        break;

        case ARM_USART_EVENT_RECEIVE_COMPLETE:
        {
            /* 正常に受信完了した場合の処理を記述 */
        }
        break;

        case ARM_USART_EVENT_TRANSFER_COMPLETE:
        {
            /* 正常に送受信完了した場合の処理を記述 */
        }
        break;

        default:
        {
            /* 通信異常が発生した場合の処理を記述 */
        }
        break;
    }

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

図 2-5 USART ドライバへのアクセス例（クロック同期スレーブモード）

```
#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);              /* USART ドライバ初期化 */
    (void)sci0Drv->PowerControl(ARM_POWER_FULL);            /* USART のモジュールストップ解除 */
    (void)sci0Drv->Control(ARM_USART_MODE_SMART_CARD |      /* スマートカードモード */
                        ARM_USART_PARITY_EVEN,              /* 偶数宇パリティ */
                        9600);                              /* 通信速度: 9600bps */
    (void)sci0Drv->Control(ARM_USART_CONTROL_TX_RX,1);      /* 送受信許可 */

    (void)sci0Drv->Send(&tx_data[0],3);                     /* 3 バイト送信開始 */

    while(1);
}

/*****************************************************************************************
* callback function
*****************************************************************************************/
static void usart_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_USART_EVENT_SEND_COMPLETE:
        {
            /* 正常に送信完了した場合の処理を記述 */
        }
        break;

        case ARM_USART_EVENT_RECEIVE_COMPLETE:
        {
            /* 正常に受信完了した場合の処理を記述 */
        }
        break;

        default:
        {
            /* 通信異常が発生した場合の処理を記述 */
        }
        break;
    }

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

図 2-6 USART ドライバへのアクセス例（スマートカードモード）

2.3 端子設定

本ドライバで使用する端子は、pin.c の R_SCI_Pinset_CHn(n=0～5,9)関数で設定、R_SCI_Pinclr_CHn 関数で解放されます。R_SCI_Pinset_CHn 関数は Control 関数で送信または受信が許可状態になったときに呼び出されます。R_SCI_Pinclr_CHn 関数は Control 関数、PowerControl 関数、または Uninitialize 関数で送受信が禁止状態になったときに呼び出されます。

使用する端子は、pin.c の R_SCI_Pinset_CHn、R_SCI_Pinclr_CHn 関数内を修正して選択してください。

2.4 通信制御および NVIC 割り込み設定

USART ドライバでは送信制御（送信データを送信バッファに書き込む処理）、受信制御（受信データを指定したバッファに格納する処理）に、デフォルトで割り込み処理を使用します。r_usart_cfg.h の送信/受信制御定義の設定値を変更することで、DMAC または DTC にて送信制御、受信制御を行うことができます。

送信/受信制御方法の設定定義を表 2-11 に、送信/受信制御方法の定義を表 2-12 に示します。

表 2-11 送信/受信制御方法の設定定義（n=0～5、9）

定義	初期値	内容
SCIn_TRANSMIT_CONTROL	SCI_USED_INTERRUPT	SCIn の送信制御（初期値：割り込み）
SCIn_RECEIVE_CONTROL	SCI_USED_INTERRUPT	SCIn の受信制御（初期値：割り込み）

表 2-12 送信/受信制御方法の定義

定義	値	内容
SCI_USED_INTERRUPT	(0)	送信/受信制御に割り込みを使用
SCI_USED_DMAC0	(1<<0)	送信/受信制御に DMAC0 を使用
SCI_USED_DMAC1	(1<<1)	送信/受信制御に DMAC1 を使用
SCI_USED_DMAC2	(1<<2)	送信/受信制御に DMAC2 を使用
SCI_USED_DMAC3	(1<<3)	送信/受信制御に DMAC3 を使用
SCI_USED_DTC	(1<<15)	送信/受信制御に DTC を使用

通信制御で使用する割り込みは、r_system_cfg.h にてネスト型ベクタ割り込みコントローラ（以下、NVIC）に登録する必要があります。詳細は「RE01 1500KB、256KB グループ CMSIS パッケージを用いた開発スタートアップガイド（r01an4660）」の「割り込み制御」を参照してください。

使用用途に対する NVIC の登録定義を表 2-13 に、NVIC への割り込み登録例を図 2-7 示します。

表 2-13　使用用途に対する NVIC の登録定義(n=0～5、9、m=0～3)

モード	使用用途	NVIC 登録定義
調歩同期	送信のみで使用	[送信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_TXI [送信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
	受信のみで使用	[受信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI [受信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI(注) SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SCIn_ERI
	送受信で使用	[送信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_TXI [送信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		[受信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI [受信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI(注) SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SCIn_ERI
クロック同期	送信のみで使用	[送信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_TXI [送信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
	送受信、または 受信のみで使用	[送信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_TXI [送信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		[受信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI [受信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI(注) SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SCIn_ERI
スマートカード	受信のみで使用	[受信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI [受信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI(注) SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SCIn_ERI
	送受信で使用	[送信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_TXI [送信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		[受信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI [受信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_SCIn_RXI(注) SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SCIn_ERI

注　DMAC を使用した場合でも、受信開始前のデータ取得時は割り込み処理を使用して受信データの破棄を行います。

```
・・・

#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 */
・・・
```

図 2-7 r_system_cfg.h での NVIC への割り込み登録例(SCI0 使用時)

2.5 マクロ／型定義

USART ドライバで、ユーザが参照可能なマクロ／型定義を Driver_USART.h、R_Driver_USART.h ファイルで定義しています。

2.5.1 USART 制御コマンド定義

USART 制御コマンドは、Control 関数の第 1 引数で使用する USART のモード、および機能の定義です。

制御コマンド定義は、機能設定、データ長設定、パリティ設定、ストップビット長設定、クロック極性(CPOL)設定、クロック位相遅れ(CPHA)設定の組み合わせで構成します。機能設定ビット（b0-b7）で USART の通信モードを設定する場合は、ほかのビット（b8-b17）でデータ長、パリティ、ストップビット長、クロック極性、クロック位相も設定してください。

USART 制御コマンド定義の構成を図 2-8 に、各機能の設定定義を表 2-14～表 2-20 に示します。

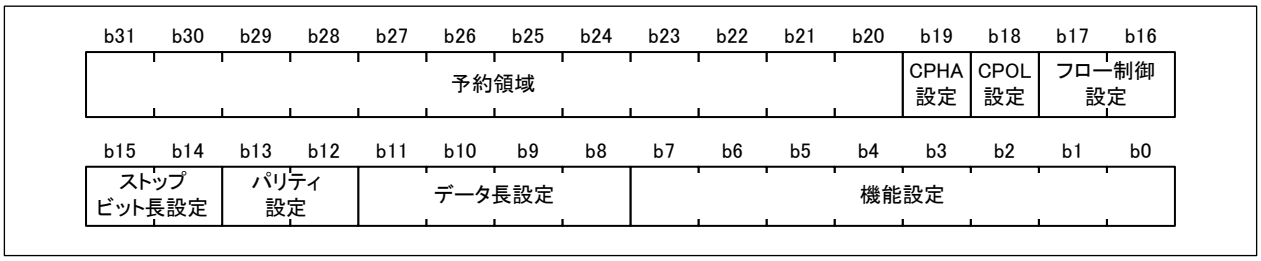


図 2-8 USART 制御コマンド定義の構成

表 2-14 USART 制御コマンド定義（機能設定定義）一覧

定義	値	内容
ARM_USART_MODE_ASYNCHRONOUS	(0x01UL << ARM_USART_CONTROL_Pos)	調歩同期モードで初期化 (注 2)
ARM_USART_MODE_SYNCHRONOUS_MASTER	(0x02UL << ARM_USART_CONTROL_Pos)	クロック同期マスタモードで初期化(注 3)
ARM_USART_MODE_SYNCHRONOUS_SLAVE	(0x03UL << ARM_USART_CONTROL_Pos)	クロック同期スレーブモードで初期化(注 3)
ARM_USART_MODE_SINGLE_WIRE	(0x04UL << ARM_USART_CONTROL_Pos)	使用禁止(注 1)
ARM_USART_MODE_IRDA	(0x05UL << ARM_USART_CONTROL_Pos)	使用禁止(注 1)
ARM_USART_MODE_SMART_CARD	(0x06UL << ARM_USART_CONTROL_Pos)	スマートカードモードで初期化(注 4)
ARM_USART_SET_DEFAULT_TX_VALUE	(0x10UL << ARM_USART_CONTROL_Pos)	デフォルトデータの設定
ARM_USART_SET_SMART_CARD_CLOCK	(0x13UL << ARM_USART_CONTROL_Pos)	スマートカードクロックの出力有無を設定
ARM_USART_CONTROL_SMART_CARD_NACK	(0x14UL << ARM_USART_CONTROL_Pos)	スマートカードモードの NACK 出力許可設定
ARM_USART_CONTROL_TX	(0x15UL << ARM_USART_CONTROL_Pos)	送信の許可/禁止設定
ARM_USART_CONTROL_RX	(0x16UL << ARM_USART_CONTROL_Pos)	受信の許可/禁止設定
ARM_USART_CONTROL_TX_RX	(0x1BUL << ARM_USART_CONTROL_Pos)	送受信の許可/禁止設定
ARM_USART_ABORT_SEND	(0x18UL << ARM_USART_CONTROL_Pos)	送信中断
ARM_USART_ABORT_RECEIVE	(0x19UL << ARM_USART_CONTROL_Pos)	受信中断
ARM_USART_ABORT_TRANSFER	(0x1AUL << ARM_USART_CONTROL_Pos)	送受信中断
ARM_USART_SET_IRDA_PULSE	(0x11UL << ARM_USART_CONTROL_Pos)	使用禁止(注 5)
ARM_USART_SET_SMART_CARD_GUARD_TIME	(0x12UL << ARM_USART_CONTROL_Pos)	使用禁止(注 5)
ARM_USART_CONTROL_BREAK	(0x17UL << ARM_USART_CONTROL_Pos)	使用禁止(注 5)

注1. USART ドライバではサポートしていません。Control 関数で本定義を指定した場合、ARM_USART_ERROR_MODE を返します。

注2. データビット長定義、パリティ機能定義、ストップビット長定義、フロー制御定義と組み合わせて設定してください。

注3. フロー制御定義、クロック極性定義、クロック位相定義と組み合わせて設定してください。

注4. パリティ機能定義と組み合わせて設定してください。

注5. USART ドライバではサポートしていません。Control 関数で本定義を指定した場合、ARM_DRIVER_ERROR を返します。

表 2-15 USART 制御コマンド（データ長設定定義）一覧

定義	値	内容
ARM_USART_DATA_BITS_7	(7UL << ARM_USART_DATA_BITS_Pos)	7 ビット長設定
ARM_USART_DATA_BITS_8	(0UL << ARM_USART_DATA_BITS_Pos)	8 ビット長設定
ARM_USART_DATA_BITS_9	(1UL << ARM_USART_DATA_BITS_Pos)	9 ビット長設定
ARM_USART_DATA_BITS_5	(5UL << ARM_USART_DATA_BITS_Pos)	使用禁止(注)
ARM_USART_DATA_BITS_6	(6UL << ARM_USART_DATA_BITS_Pos)	使用禁止(注)

注　USART ドライバではサポートしていません。Control 関数で本定義を指定した場合、ARM_USART_ERROR_DATA_BITS を返します。

表 2-16 USART 制御コマンド（パリティ設定定義）一覧

定義	値	内容
ARM_USART_PARITY_NONE,	(0UL << ARM_USART_PARITY_Pos)	パリティなし設定
ARM_USART_PARITY_EVEN	(1UL << ARM_USART_PARITY_Pos)	偶数パリティ設定
ARM_USART_PARITY_ODD	(2UL << ARM_USART_PARITY_Pos)	奇数パリティ設定

表 2-17 USART 制御コマンド（ストップビット長設定定義）一覧

定義	値	内容
ARM_USART_STOP_BITS_1	(0UL << ARM_USART_STOP_BITS_Pos)	1 ストップビット長設定
ARM_USART_STOP_BITS_2	(1UL << ARM_USART_STOP_BITS_Pos)	2 ストップビット長設定
ARM_USART_STOP_BITS_1_5	(2UL << ARM_USART_STOP_BITS_Pos)	使用禁止(注)
ARM_USART_STOP_BITS_0_5	(3UL << ARM_USART_STOP_BITS_Pos)	使用禁止(注)

注　USART ドライバではサポートしていません。Control 関数で本定義を指定した場合、ARM_USART_ERROR_STOP_BITS を返します。

表 2-18 USART 制御コマンド（フロー制御設定定義）一覧

定義	値	内容
ARM_USART_FLOW_CONTROL_NONE,	(0UL << ARM_USART_FLOW_CONTROL_Pos)	フロー制御なし設定
ARM_USART_FLOW_CONTROL_RTS	(1UL << ARM_USART_FLOW_CONTROL_Pos)	RTS 制御設定
ARM_USART_FLOW_CONTROL_CTS	(2UL << ARM_USART_FLOW_CONTROL_Pos)	CTS 制御設定
ARM_USART_FLOW_CONTROL_RTS_CTS	(3UL << ARM_USART_FLOW_CONTROL_Pos)	使用禁止(注)

注　USART ドライバではサポートしていません。Control 関数で本定義を指定した場合、ARM_USART_ERROR_FLOW_CONTROL を返します。

表 2-19 USART 制御コマンド（クロック極性定義）一覧

定義	値	内容
ARM_USART_CPOL0	(0UL << ARM_USART_CPOL_Pos)	クロック極性反転なし設定
ARM_USART_CPOL1	(1UL << ARM_USART_CPOL_Pos)	クロック極性反転あり設定

表 2-20 USART 制御コマンド（クロック位相遅れ定義）一覧

定義	値	内容
ARM_USART_CPHA0	(0UL << ARM_USART_CPHA_Pos)	クロック位相遅れなし設定
ARM_USART_CPHA1	(1UL << ARM_USART_CPHA_Pos)	クロック位相遅れあり設定

2.5.2 USART 特定のエラーコード定義

USART 特定のエラーコード定義です。

表 2-21 USART 特定エラーコード定義一覧

定義	値	内容
ARM_USART_ERROR_MODE	(ARM_DRIVER_ERROR_SPECIFIC - 1)	指定モードはサポートされていません
ARM_USART_ERROR_BAUDRATE	(ARM_DRIVER_ERROR_SPECIFIC - 2)	指定されたボーレートはサポートされていません
ARM_USART_ERROR_DATA_BITS	(ARM_DRIVER_ERROR_SPECIFIC - 3)	指定されたビット長はサポートされていません
ARM_USART_ERROR_PARITY	(ARM_DRIVER_ERROR_SPECIFIC - 4)	指定されたパリティはサポートされていません
ARM_USART_ERROR_STOP_BITS	(ARM_DRIVER_ERROR_SPECIFIC - 5)	指定されたストップビット長はサポートされていません
ARM_USART_ERROR_FLOW_CONTROL	(ARM_DRIVER_ERROR_SPECIFIC - 6)	指定されたフロー制御はサポートされていません
ARM_USART_ERROR_CPOL	(ARM_DRIVER_ERROR_SPECIFIC - 7)	未使用(注)
ARM_USART_ERROR_CPHA	(ARM_DRIVER_ERROR_SPECIFIC - 8)	未使用(注)

注　USART ドライバではクロック極性反転、クロック位相遅れをサポートしているため、本エラーを返しません

2.5.3 モデム制御定義

ARM_USART_SetModemControl 関数で使用するモデム制御用定義です。

表 2-22 モデム制御定義一覧

定義	値	内容
ARM_USART_RTS_CLEAR	(0)	RTS 出力を非アクティブ(“H”)にします
ARM_USART_RTS_SET	(1)	RTS 出力をアクティブ(“L”)にします
ARM_USART_DTR_CLEAR	(2)	使用禁止(注)
ARM_USART_DTR_SET	(3)	使用禁止(注)

注　本ドライバではサポートしていません。

2.5.4 USART イベントコード定義

コールバック関数で通知されるイベント定義です。複数イベントが同時に発生した場合は、OR 結合した値を通知します。コールバック関数でのイベントコードの判定例は、各モードでの API アクセス例（「2.2 ドライバ API」の図 2-3～図 2-6）を参照してください。

表 2-23 USART イベントコード一覧

定義	値	内容
ARM_USART_EVENT_SEND_COMPLETE	(1UL << 0)	送信完了しました
ARM_USART_EVENT_RECEIVE_COMPLETE	(1UL << 1)	受信完了しました
ARM_USART_EVENT_TRANSFER_COMPLETE	(1UL << 2)	送受信完了しました
ARM_USART_EVENT_TX_COMPLETE	(1UL << 3)	未使用
ARM_USART_EVENT_TX_UNDERFLOW	(1UL << 4)	未使用
ARM_USART_EVENT_RX_OVERFLOW	(1UL << 5)	受信オーバフローが発生しました
ARM_USART_EVENT_RX_TIMEOUT	(1UL << 6)	未使用
ARM_USART_EVENT_RX_BREAK	(1UL << 7)	未使用
ARM_USART_EVENT_RX_FRAMING_ERROR	(1UL << 8)	フレーミングエラーが発生しました
ARM_USART_EVENT_RX_PARITY_ERROR	(1UL << 9)	パリティエラーが発生しました
ARM_USART_EVENT_CTS	(1UL << 10)	未使用
ARM_USART_EVENT_DSR	(1UL << 11)	未使用
ARM_USART_EVENT_DCD	(1UL << 12)	未使用
ARM_USART_EVENT_RI	(1UL << 13)	未使用

2.6 構造体定義

USART ドライバでは、ユーザが参照可能な構造体定義を Driver_USART.h ファイルで定義しています。

2.6.1 ARM_USART_STATUS 構造体

GetStatus 関数で USART の状態を返すときに使用する構造体です。

表 2-24 ARM_USART_STATUS 構造体

要素名	型	内容
tx_busy	uint32_t:1	送信状態を示します 0: 送信待機中 1: 送信中(ビジー)
rx_busy	uint32_t:1	受信状態を示します 0: 受信待機中 1: 受信中(ビジー)
tx_underflow	uint32_t:1	未使用(0 固定)
rx_overflow	uint32_t:1	受信オーバフロー発生状態を示します 0: 受信オーバフロー未発生 1: 受信オーバフロー発生
rx_break	uint32_t:1	未使用(0 固定)
rx_framing_error	uint32_t:1	フレーミングエラー発生状態を示します 0: フレーミングエラー未発生 1: フレーミングエラー発生
rx_parity_error	uint32_t:1	パリティエラー発生状態を示します 0: パリティエラー未発生 1: パリティエラー発生
reserved	uint32_t:25	予約領域

2.6.2 ARM_USART_MODEM_STATUS 構造体

GetModemStatus 関数でモデムの状態を返すときに使用する構造体です。

表 2-25 ARM_USART_MODEM_STATUS 構造体

要素名	型	内容
cts	uint32_t:1	CTS 状態を示します 0: CTS 非アクティブ状態 1: CTS アクティブ状態
dsr	uint32_t:1	未使用(0 固定)
dcd	uint32_t:1	未使用(0 固定)
ri	uint32_t:1	未使用(0 固定)
reserved	uint32_t:28	予約領域

2.6.3 ARM_USART_CAPABILITIES 構造体

GetCapabilities 関数で USART の機能を返すときに使用する構造体です。

表 2-26 ARM_USART_CAPABILITIES 構造体

要素名	型	内容	値
asynchronous	uint32_t:1	調歩同期モードの有効/無効	1(有効)
synchronous_master	uint32_t:1	クロック同期マスタモードの有効/無効	1(有効)
synchronous_slave	uint32_t:1	クロック同期スレーブモードの有効/無効	1(有効)
single_wire	uint32_t:1	シングルワイヤモードの有効/無効	0(無効)
irda	uint32_t:1	IRDA モードの有効/無効	0(無効)
smart_card	uint32_t:1	スマートカードモードの有効/無効	1(有効)
smart_card_clock	uint32_t:1	スマートカードクロック出力の有効/無効	1(有効)
flow_control_rts	uint32_t:1	RTS フロー制御の有効/無効	1(有効)
flow_control_cts	uint32_t:1	CTS フロー制御の有効/無効	1(有効)
event_tx_complete	uint32_t:1	送信完了イベントの有効/無効	0(無効)
event_rx_timeout	uint32_t:1	受信タイムアウトイベントの有効/無効	0(無効)
rts	uint32_t:1	RTS ラインの有効/無効	1(有効)
cts	uint32_t:1	CTS ラインの有効/無効	1(有効)
dtr	uint32_t:1	DTR ラインの有効/無効	0(無効)
dsr	uint32_t:1	DSR ラインの有効/無効	0(無効)
dcd	uint32_t:1	DCD ラインの有効/無効	0(無効)
ri	uint32_t:1	RI ラインの有効/無効	0(無効)
event_cts	uint32_t:1	CTS イベントの有効/無効	0(無効)
event_dsr	uint32_t:1	DSR イベントの有効/無効	0(無効)
event_dcd	uint32_t:1	DCD イベントの有効/無効	0(無効)
event_ri	uint32_t:1	RI イベントの有効/無効	0(無効)
reserved	uint32_t:11	予約領域	-

2.7 状態遷移

USART ドライバの状態遷移図を図 2-9 に、各状態でのイベント動作を表 2-27 に示します。

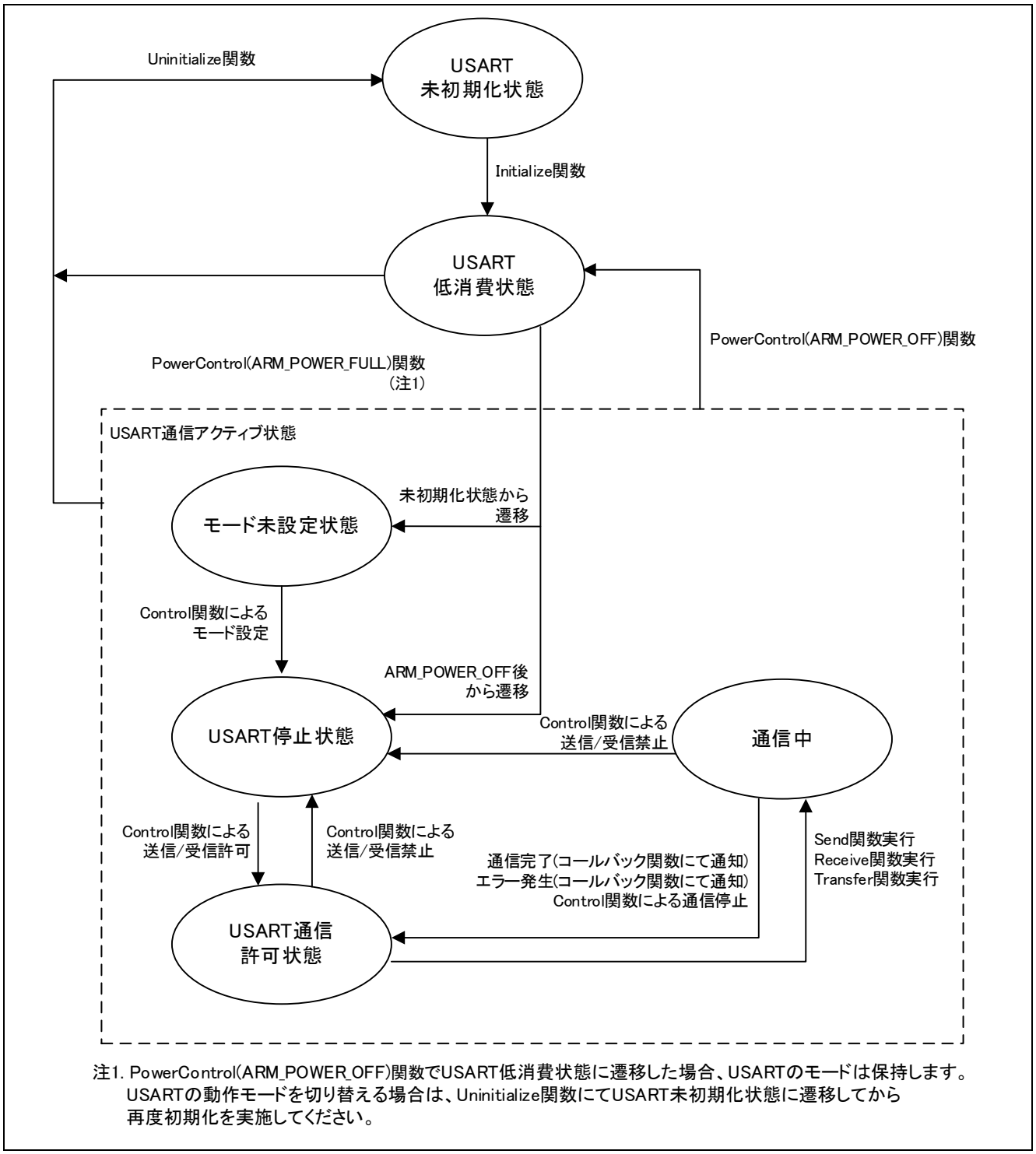


図 2-9 USART ドライバの状態遷移

表 2-27 USART ドライバ状態でのイベント動作(注 1)

状態	概要	イベント	アクション
USART 未初期化状態	リセット解除後の USART ドライバの状態です	Initialize 関数の実行	USART 低消費状態に遷移
USART 低消費状態	SCI モジュールにクロックが供給されていない状態です	Uninitialize 関数の実行	USART 未初期化状態に遷移
		PowerControl(ARM_POWER_FULL)関数の実行	モード未設定状態、または USART 停止状態に遷移 (注 2)
モード未設定状態	USART モードが未設定の状態です	Control(ARM_USART_MODE_XXX)関数の実行(注 3)	USART 停止状態に遷移
		Uninitialize 関数の実行	USART 未初期化状態に遷移
		PowerControl(ARM_POWER_OFF)関数の実行	USART 低消費状態に遷移
USART 停止状態	通信停止状態です	Uninitialize 関数の実行	USART 未初期化状態に遷移
		PowerControl(ARM_POWER_OFF)関数の実行	USART 低消費状態に遷移
		Control 関数による送信/受信許可	USART 通信許可状態に遷移
		SetModemControl 関数の実行	RTS 端子制御(注 4)
USART 通信許可状態	通信待ち状態です	Uninitialize 関数の実行	USART 未初期化状態に遷移
		PowerControl(ARM_POWER_OFF)関数の実行	USART 低消費状態に遷移
		Control 関数による送信/受信禁止	USART 停止状態に遷移
		Send 関数の実行	通信中状態に遷移(送信開始)
		Receive 関数の実行	通信中状態に遷移(受信開始)
		Transfer 関数の実行	通信中状態に遷移(送受信開始)
		SetModemControl 関数の実行	RTS 端子制御(注 4)
通信中状態	通信状態です	Uninitialize 関数の実行	USART 未初期化状態に遷移
		PowerControl(ARM_POWER_OFF)関数の実行	USART 低消費状態に遷移
		通信の完了	USART 通信許可状態に遷移し、コールバック関数を呼び出します(注 5)
		エラー発生	USART 通信許可状態に遷移し、コールバック関数を呼び出します(注 5)
		Control 関数による送信/受信禁止	USART 停止状態に遷移
		Control(ARM_USART_ABORT_XXX)関数の実行	通信を中断し、USART 通信許可状態に遷移します
		SetModemControl 関数の実行	RTS 端子制御(注 4)

注1. GetVersion、GetCapabilities、GetTxCount、GetRxCount、GetStatus、GetModemStatus 関数はすべての状態で実行可能です。

注2. USART 未初期化状態から USART モードを設定していない場合は、モード未設定状態に遷移します。

注3. XXX は以下のいずれか
ASYNCHRONOUS : 調歩同期モード
SYNCHRONOUS_MASTER : クロック同期マスタモード
SYNCHRONOUS_SLAVE : クロック同期スレーブモード
SMART_CARD : スマートカードモード

注4. r_usart_cfg.h で RTS 端子を設定し、かつモード設定時にハードウェアによる RTS 機能を無効に設定した場合のみ有効です。

注5. Initialize 関数実行時にコールバック関数を指定していた場合のみ、コールバック関数を呼び出します。

3. ドライバ動作説明

USART ドライバは調歩同期通信、クロック同期通信、スマートカード通信機能を実現します。本章では各モードで USART ドライバを設定する手順について示します。

3.1 調歩同期モード

3.1.1 調歩同期モード初期設定手順

調歩同期モードの初期設定手順を図 3-1 に示します。

送信・受信を許可にする場合、r_system_cfg.h にて使用する割り込みを NVIC に登録してください。詳細は「2.4 通信制御」を参照してください。

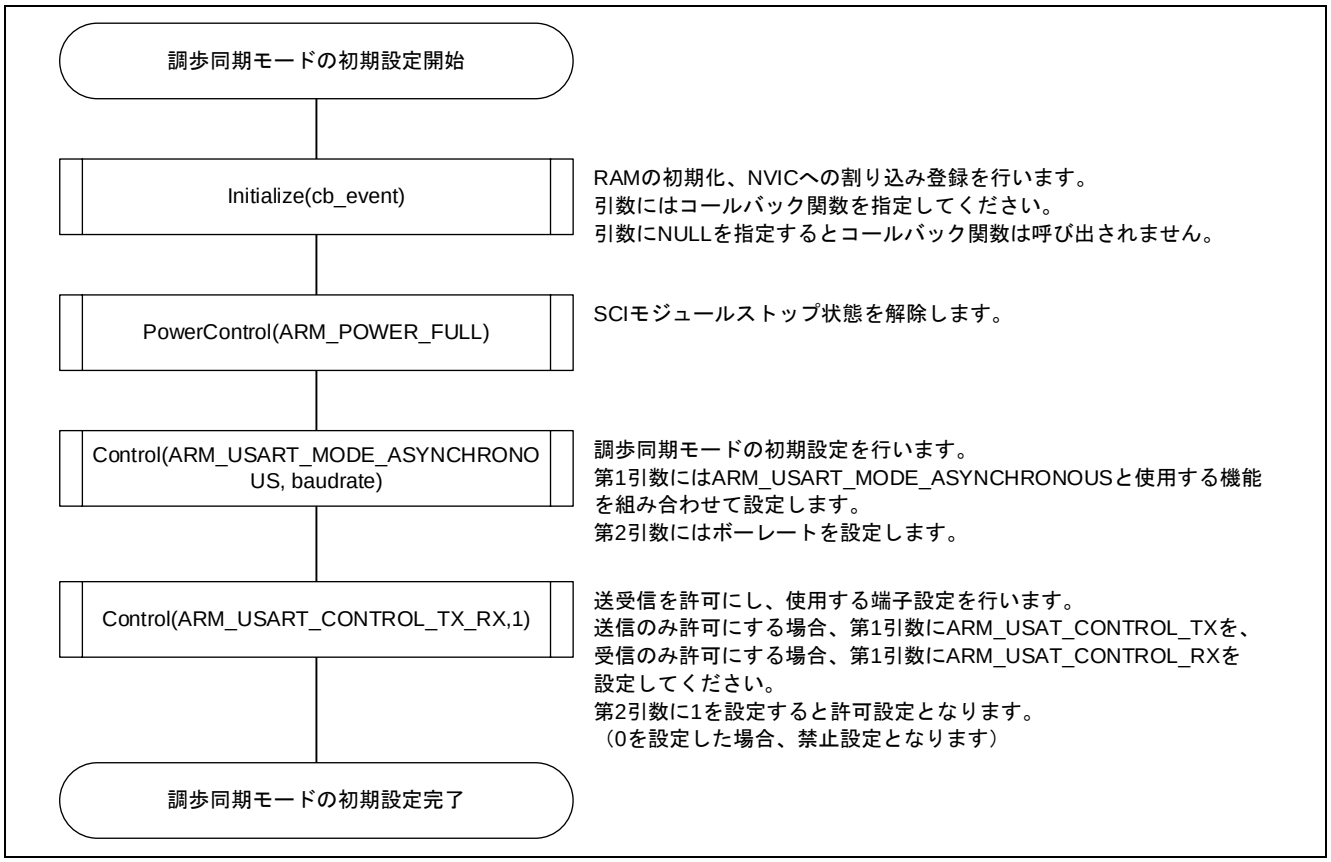


図 3-1 調歩同期モードの初期化手順

3.1.2 調歩同期モードでの送信処理

調歩同期モードで送信を行う手順を図 3-2 に示します。

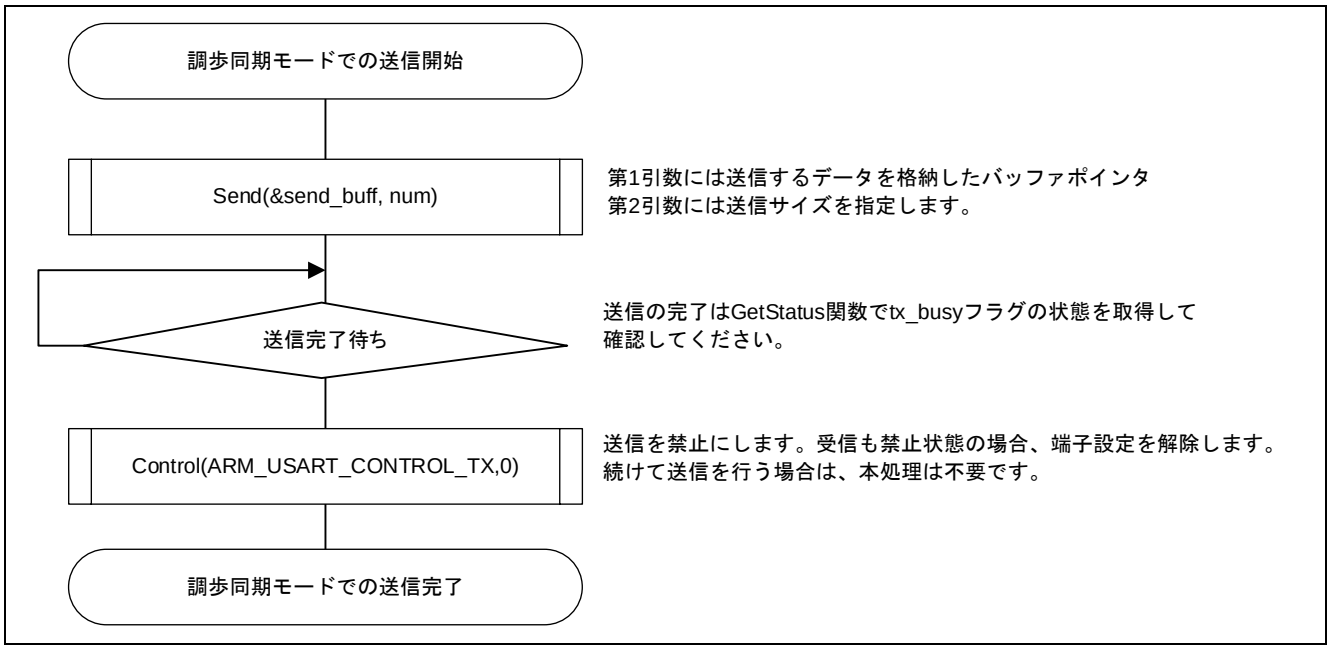


図 3-2 調歩同期モードでの送信手順

コールバック関数を設定していた場合、送信が完了すると ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数が呼び出されます。

調歩同期による送信処理は、通信制御の設定が割り込み、または DMAC、または DTC にて処理方法が異なります。図 3-3 に通信制御が割り込みの場合の動作を、図 3-4 に通信制御が DMAC の場合の動作を、図 3-5 に通信制御が DTC の場合の動作を示します。

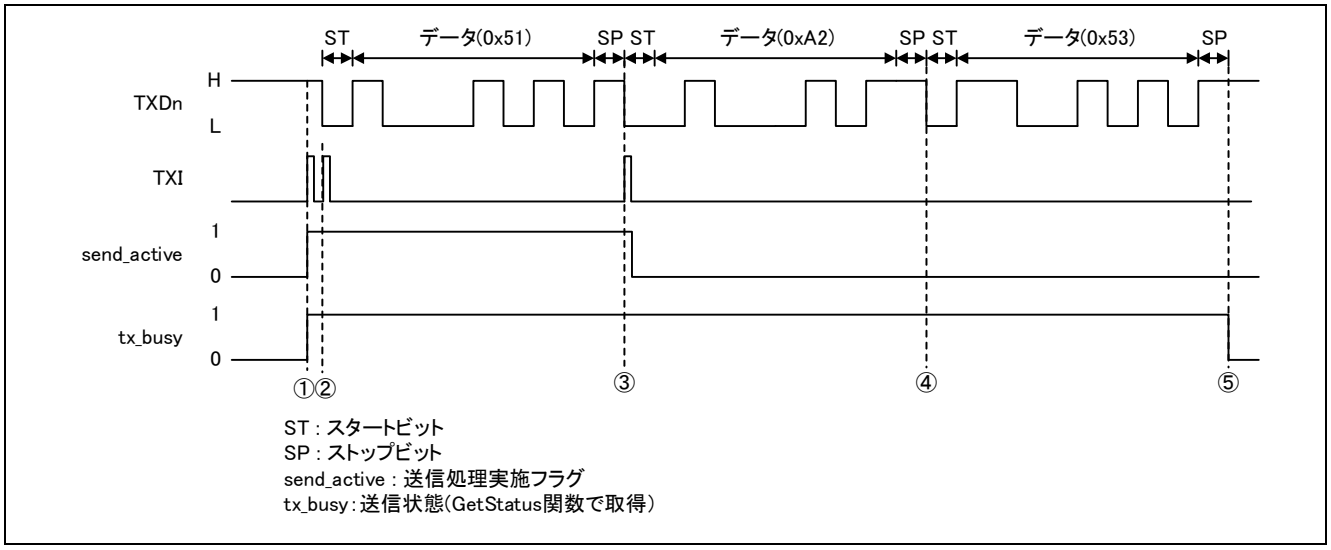


図 3-3 割り込み制御による送信動作（3 バイト送信）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)になります。また、TXI 割り込みが発生し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② 2 回目の TXI 割り込みにて 2 バイト目の送信データを TDR レジスタに書き込みます。

③ 最終データ書き込み後の TXI 割り込みで、TXI 割り込みを禁止にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 2 バイト目のデータが出力されたのち、③で書き込んだ最終データが出力されます。(注)

⑤ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

注 send_active フラグが”0”の状態、かつ送信バッファに値がある状態（③～④の期間）で Send 関数を実行した場合、send_active フラグが”1”になったのち、Sned 関数を終了します。送信バッファが空になった時点（④のタイミング）で TXI 割り込みが発生し、1 バイト目のデータを TDR レジスタに書き込みます。

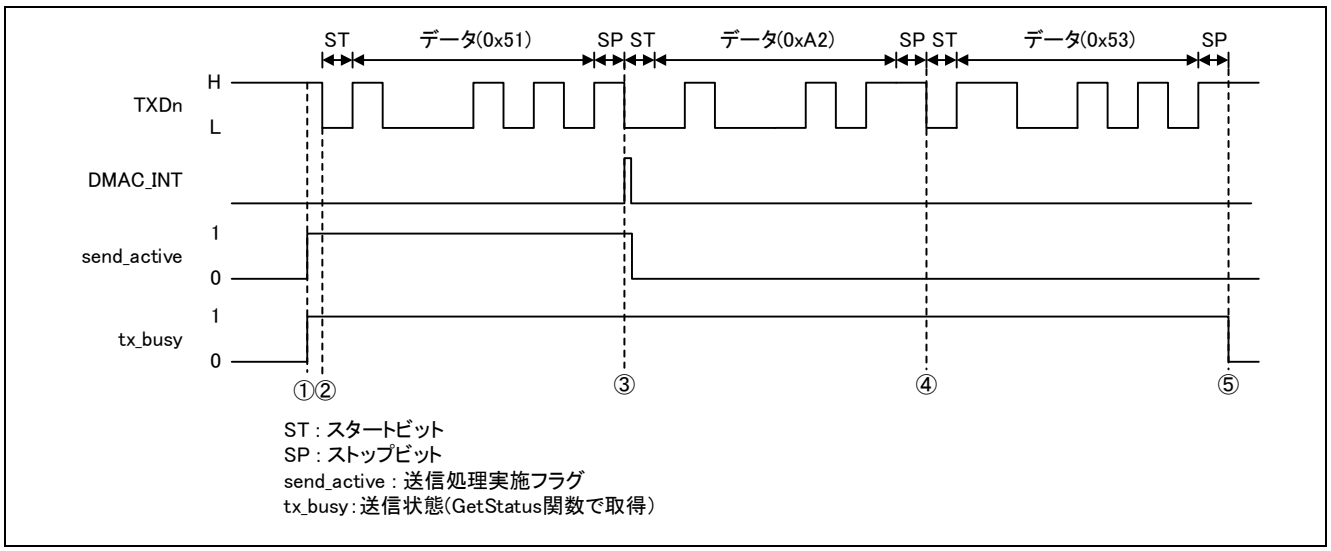


図 3-4 DMAC 制御による送信動作（3 バイト送信）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に設定、DMAC の転送要因に TXI 割り込みを設定し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② DMA 転送にて 2 バイト目以降のデータが送信データレジスタ（TDR）に転送されます。

③ DMA 転送にて最終データが転送されると、DMAC 転送完了割り込みが発生し、send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 2 バイト目のデータが出力されたのち、③で書き込んだ最終データが出力されます。(注)

⑤ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

注 tx_busy フラグが”0”の状態、かつ送信バッファに値がある状態（③～④の期間）で Send 関数を実行した場合、送信バッファが空になるまで（④のタイミング）待ったのち、①の処理を実行します。
待ち時間は r_usart_cfg.h の SCI_CHECK_TDRE_TIMEOUT 定義の値で変更できます。指定した時間、送信バッファが空にならなかった場合、Send 関数は ARM_DRIVER_ERROR_BUSY を返します。

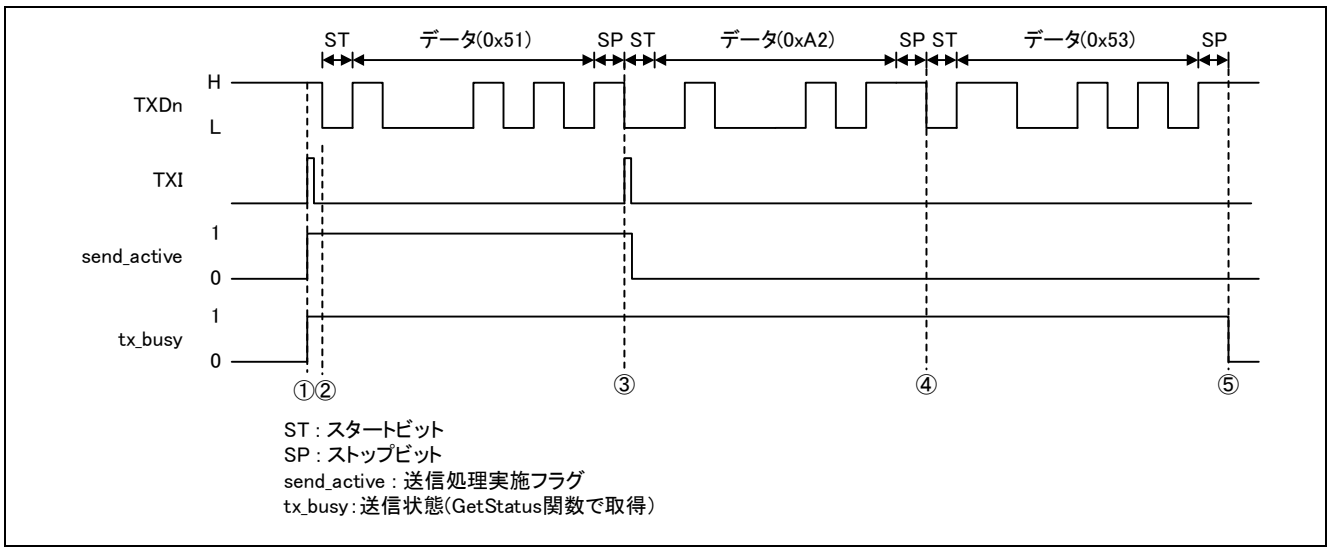


図 3-5 DTC 制御による送信動作（3 バイト送信）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に設定、DTC 転送要因に TXI 割り込みを設定し、TXI 割り込みを許可にします。このとき、TXI 割り込みが発生し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② DMA 転送にて 2 バイト目以降のデータが送信データレジスタ（TDR）に転送されます。

③ DMA 転送にて最終データが転送されると、TXI 割り込みが発生し、send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 2 バイト目のデータが出力されたのち、③で書き込んだ最終データが出力されます。(注)

⑤ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

注 tx_busy フラグが”0”の状態、かつ送信バッファに値がある状態（③～④の期間）で Send 関数を実行した場合、送信バッファが空になるまで（④のタイミング）待ったのち、①の処理を実行します。
待ち時間は r_usart_cfg.h の SCI_CHECK_TDRE_TIMEOUT 定義の値で変更できます。指定した時間、送信バッファが空にならなかった場合、Send 関数は ARM_DRIVER_ERROR_BUSY を返します。

3.1.3 調歩同期モードでの受信処理

調歩同期モードで受信を行う手順を図 3-6 に示します。

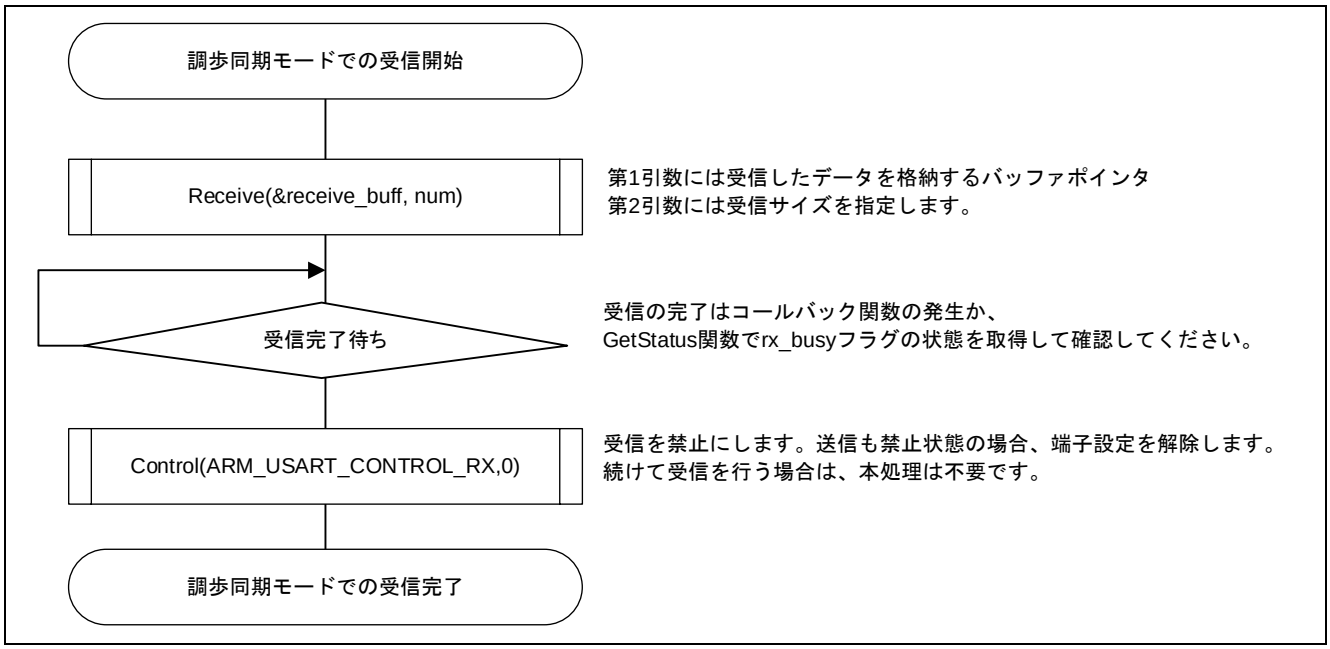


図 3-6 調歩同期モードでの受信手順

コールバック関数を設定していた場合、受信が完了すると ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数が呼び出されます。

また、受信エラーが発生した場合、エラーイベント情報を引数にコールバック関数が呼び出され、受信処理を完了します。調歩同期の受信処理で発生するエラーイベント情報を表 3-1 に示します。

表 3-1 調歩同期の受信処理で発生するエラーイベント情報

エラーイベント情報(注)	内容
ARM_USART_EVENT_RX_OVERFLOW	オーバランエラー発生
ARM_USART_EVENT_RX_FRAMING_ERROR	フレーミングエラー発生
ARM_USART_EVENT_RX_PARITY_ERROR	パリティエラー発生

注　複数のエラーが発生した場合、OR 結合されたエラーイベント情報を引数にコールバック関数が呼び出されます。

調歩同期による受信処理は、通信制御の設定が割り込み、または DMAC、または DTC にて処理方法が異なります。図 3-7 に通信制御が割り込みの場合の動作を、図 3-8 に通信制御が DMAC の場合の動作を、図 3-9 に通信制御が DTC の場合の動作を示します。

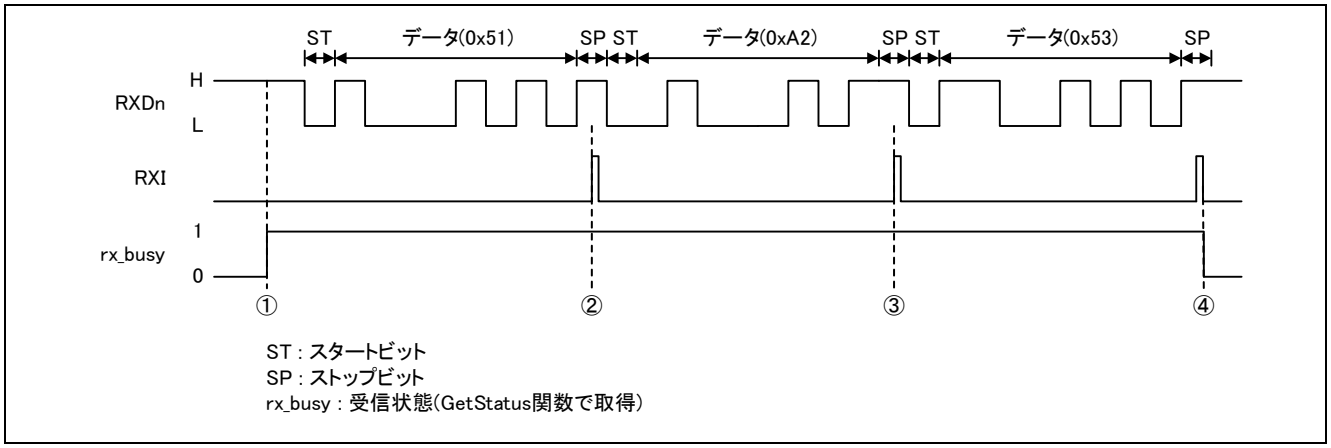


図 3-7 割り込み制御による受信動作（3 バイト受信）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)になります。

② RXDn 端子からデータを受信すると、RXI 割り込みが発生します。RXI 割り込みにて受信データレジスタ（RDR）から受信データを指定されたバッファに読み出します。

③ 1 バイト受信ごとに RXI 割り込みが発生し、RDR レジスタから受信データを読み出します。

④ 最終データ読み出し時の RXI 割り込みで、rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注　受信エラーが発生した場合、ERI 割り込みが発生し、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

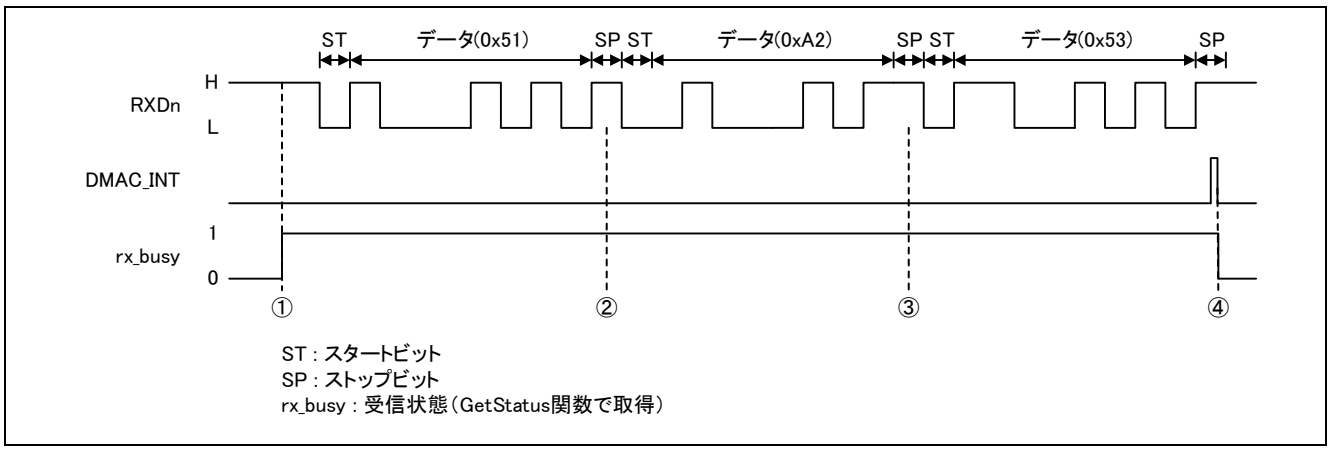


図 3-8 DMAC 制御による受信動作（3 バイト受信）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)になります。

② RXDn 端子からデータを受信すると、DMA 転送にて受信データレジスタ（RDR）から受信データを指定されたバッファに転送します。

③ 1 バイト受信ごとに DMA 転送が発生し、RDR レジスタから受信データを転送します。

④ 最終データの転送完了後、DMAC 転送完了割り込みが発生します。割り込み処理にて rx_busy フラグを”0”(受信待ち状態)にし、コールバック関数が登録されている場合、ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注　受信エラーが発生した場合、ERI 割り込みが発生し、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

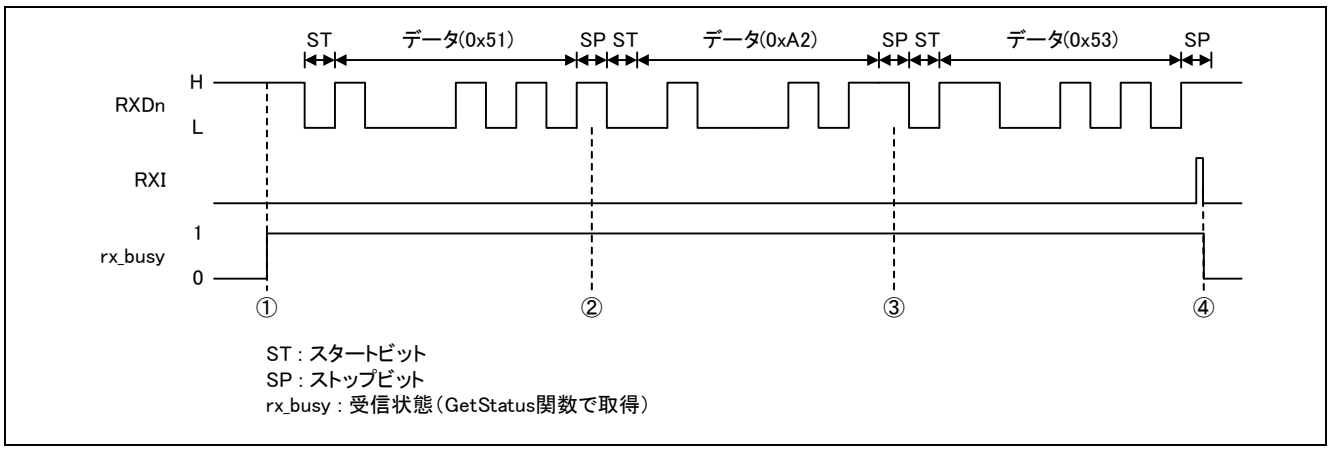


図 3-9 DTC 制御による受信動作（3 バイト受信）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)になります。

② RXDn 端子からデータを受信すると、DMA 転送にて受信データレジスタ（RDR）から受信データを指定されたバッファに転送します。

③ 1 バイト受信ごとに DMA 転送が発生し、RDR レジスタから受信データを転送します。

④ 最終データの転送完了後、RXI 割り込みが発生します。割り込み処理にて rx_busy フラグを”0”(受信待ち状態)にし、コールバック関数が登録されている場合、ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注　受信エラーが発生した場合、ERI 割り込みが発生し、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

3.2 クロック同期マスタモード

3.2.1 クロック同期マスタモード初期設定手順

クロック同期マスタモードの初期設定手順を図 3-10 に示します。

送信・受信を許可にする場合、r_system_cfg.h にて使用する割り込みを NVIC に登録してください。詳細は「2.4 通信制御」を参照してください。

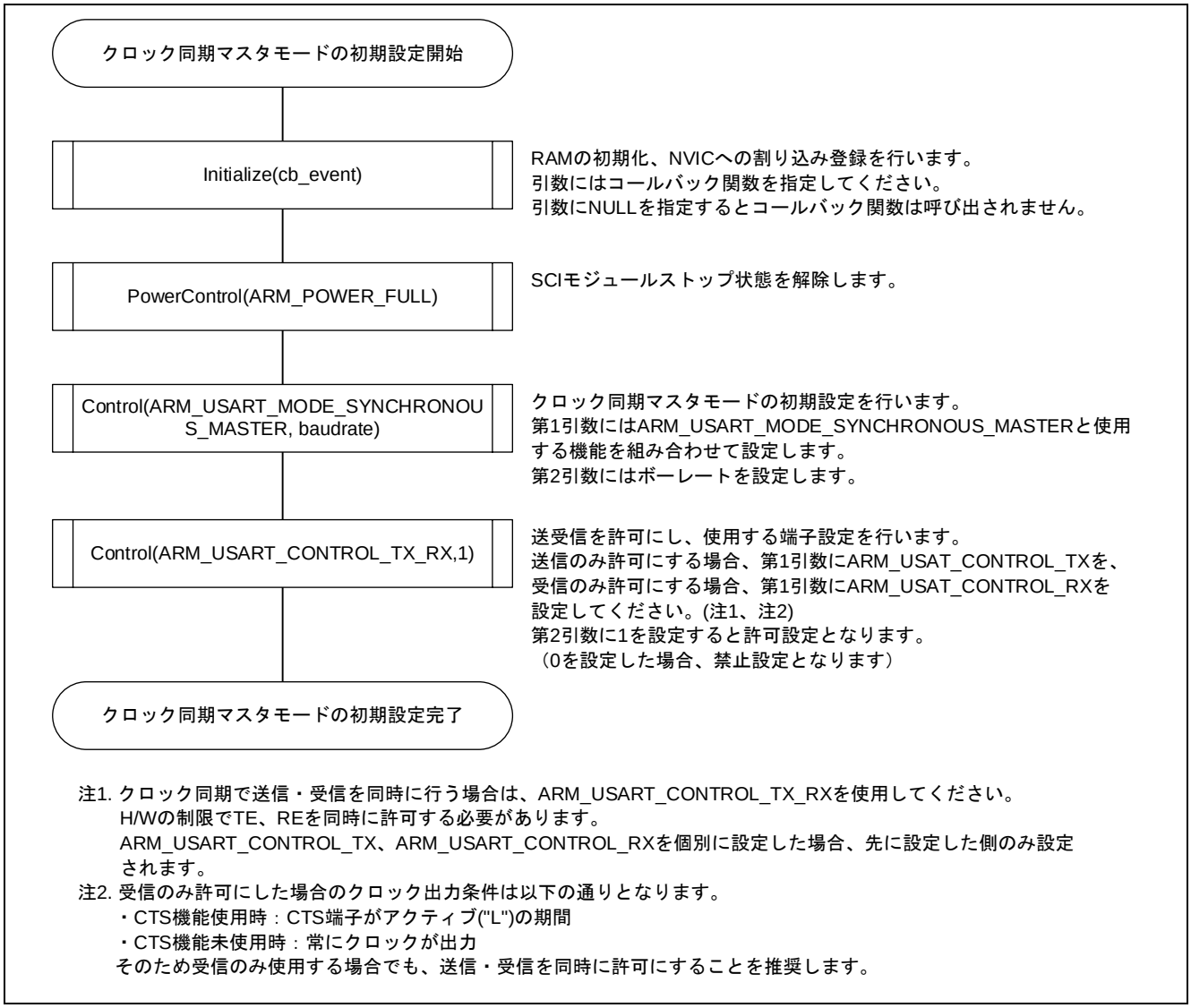


図 3-10 クロック同期マスタモードの初期化手順

3.2.2 クロック同期マスタモードでの送信処理

クロック同期マスタモードで送信を行う手順を図 3-11 に示します。

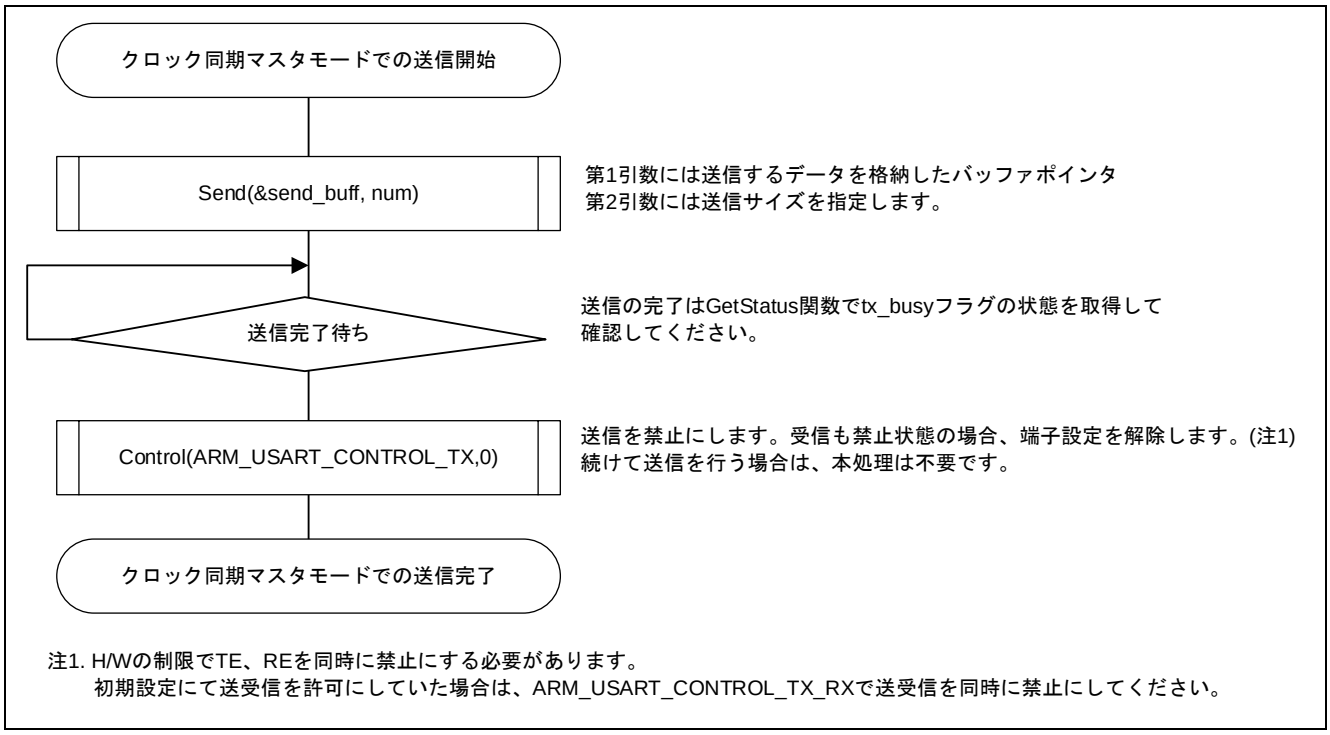


図 3-11 クロック同期マスタモードでの送信手順

コールバック関数を設定していた場合、送信が完了すると ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数が呼び出されます。

クロック同期マスタモードによる送信処理は、通信制御の設定が割り込み、または DMAC、または DTC にて処理方法が異なります。図 3-12 に通信制御が割り込みの場合の動作を、図 3-13 に通信制御が DMAC の場合の動作を、図 3-14 に通信制御が DTC の場合の動作を示します。

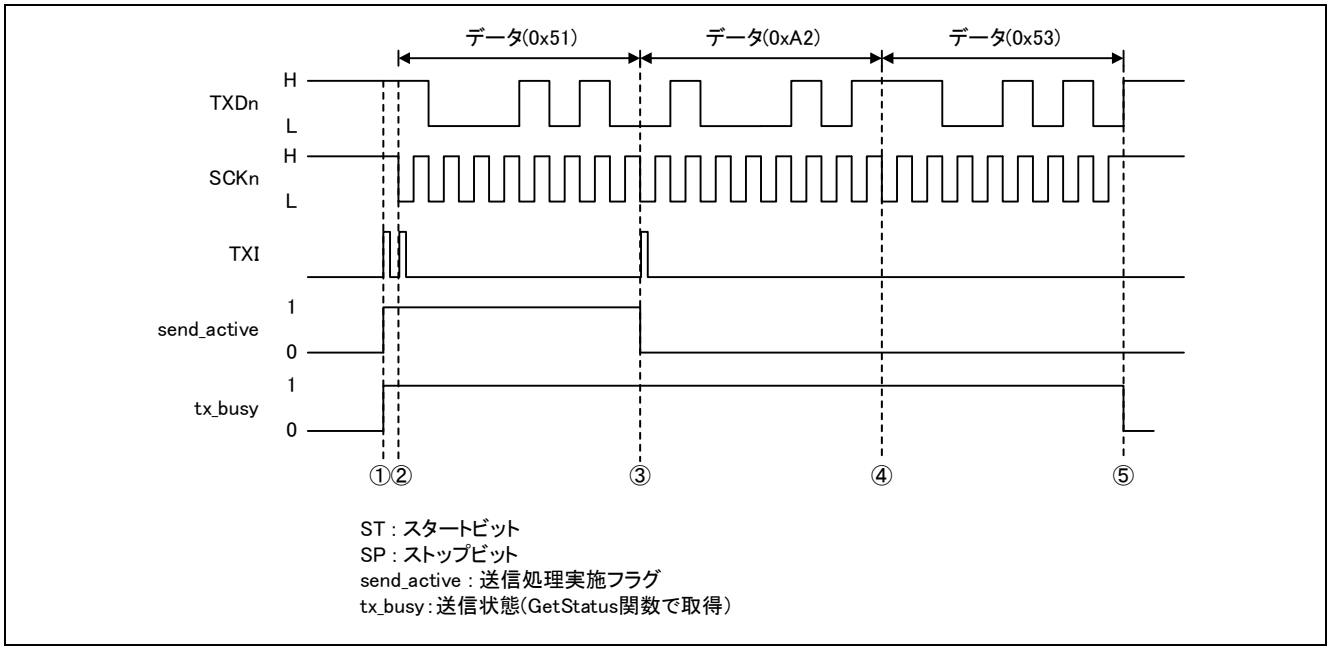


図 3-12 割り込み制御による送信動作（3 バイト送信）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)になります。また、TXI 割り込みが発生し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② 2 回目の TXI 割り込みにて 2 バイト目の送信データを TDR レジスタに書き込みます。

③ 最終データ書き込み後の TXI 割り込みで、TXI 割り込みを禁止にし、send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 2 バイト目のデータが出力されたのち、③で書き込んだ最終データが出力されます。(注)

⑤ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

注 tx_busy フラグが”0”の状態、かつ送信バッファに値がある状態（③～④の期間）で Send 関数を実行した場合、tx_busy フラグが”1”にして、Sned 関数を終了します。送信バッファが空になった時点（④のタイミング）で TXI 割り込みが発生し、1 バイト目のデータを TDR レジスタに書き込みます。

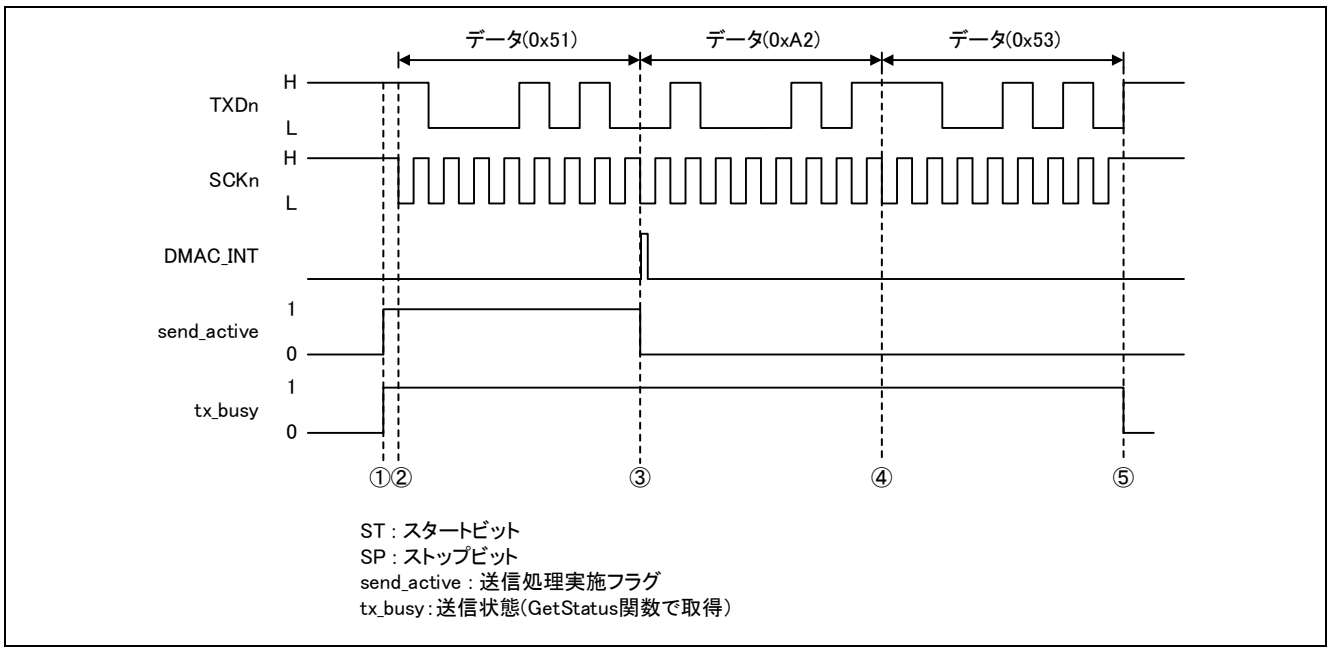


図 3-13 DMAC 制御による送信動作（3 バイト送信）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に設定、DMAC の転送要因に TXI 割り込みを設定し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② DMA 転送にて 2 バイト目以降のデータが送信データレジスタ（TDR）に転送されます。

③ DMA 転送にて最終データが転送されると、DMAC 転送完了割り込みが発生し、send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 2 バイト目のデータが出力されたのち、③で書き込んだ最終データが出力されます。(注)

⑤ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

注 tx_busy フラグが”0”の状態、かつ送信バッファに値がある状態（③～④の期間）で Send 関数を実行した場合、送信バッファが空になるまで（④のタイミング）待ったのち、①の処理を実行します。
待ち時間は r_usart_cfg.h の SCI_CHECK_TDRE_TIMEOUT 定義の値で変更できます。指定した時間、送信バッファが空にならなかった場合、Send 関数は ARM_DRIVER_ERROR_BUSY を返します。

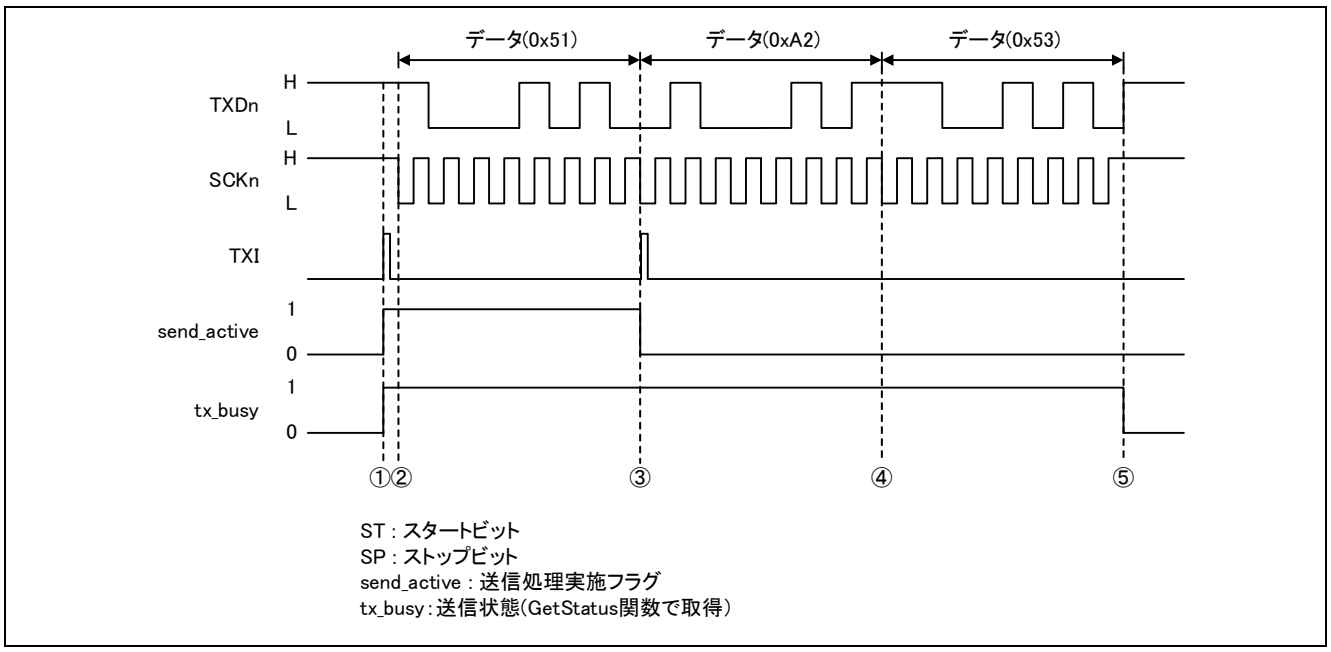


図 3-14 DTC 制御による送信動作（3 バイト送信）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に設定、DTC 転送要因に TXI 割り込みを設定し、TXI 割り込みを許可にします。このとき、TXI 割り込みが発生し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② DMA 転送にて 2 バイト目以降のデータが送信データレジスタ（TDR）に転送されます。

③ DMA 転送にて最終データが転送されると、TXI 割り込みが発生し、send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、
ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 2 バイト目のデータが出力されたのち、③で書き込んだ最終データが出力されます。(注)

⑤ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

注 tx_busy フラグが”0”の状態、かつ送信バッファに値がある状態（③～④の期間）で Send 関数を実行した場合、送信バッファが空になるまで（④のタイミング）待ったのち、①の処理を実行します。
待ち時間は r_usart_cfg.h の SCI_CHECK_TDRE_TIMEOUT 定義の値で変更できます。指定した時間、送信バッファが空にならなかった場合、Send 関数は ARM_DRIVER_ERROR_BUSY を返します。

3.2.3 クロック同期マスタモードでの受信処理

クロック同期マスタモードで受信を行う手順を図 3-15 に示します。

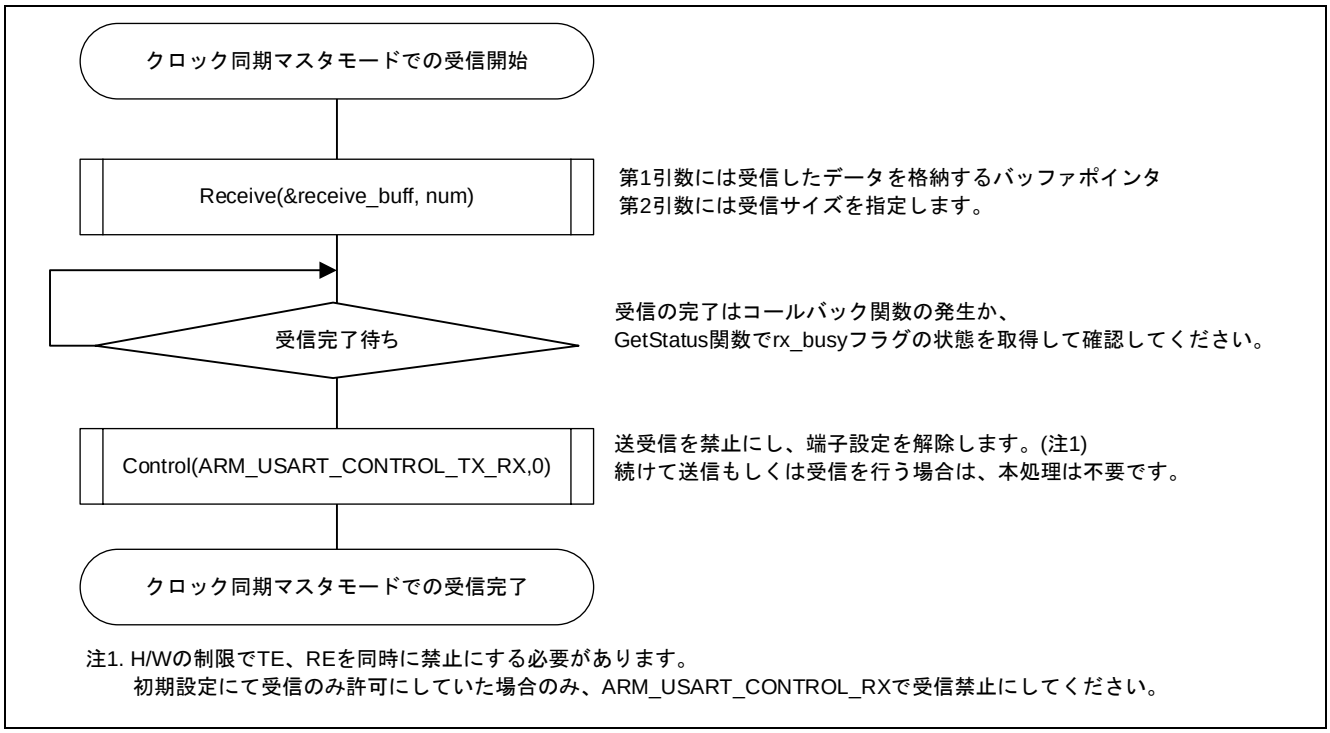


図 3-15 クロック同期マスタモードでの受信手順

コールバック関数を設定していた場合、受信が完了すると ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数が呼び出されます。

また、受信エラーが発生した場合、エラーイベント情報を引数にコールバック関数が呼び出され、受信処理を完了します。クロック同期マスタモードの受信処理で発生するエラーイベント情報を表 3-2 に示します。

表 3-2 クロック同期マスタモードの受信処理で発生するエラーイベント情報

エラーイベント情報	内容
ARM_USART_EVENT_RX_OVERFLOW	オーバランエラー発生

クロック同期マスタモードによる受信処理は、通信制御の設定が割り込み、または DMAC、または DTC にて処理方法が異なります。また、送信許可状態の場合、クロック出力を行うため、ダミーデータを送信バッファに書き込みます。出力するダミーデータは ARM_USART_SET_DEFAULT_TX_VALUE コマンドにて変更できます。

図 3-16 に通信制御が割り込みの場合の動作を、図 3-17 に通信制御が DMAC の場合の動作を、図 3-18 に通信制御が DTC の場合の動作を示します。

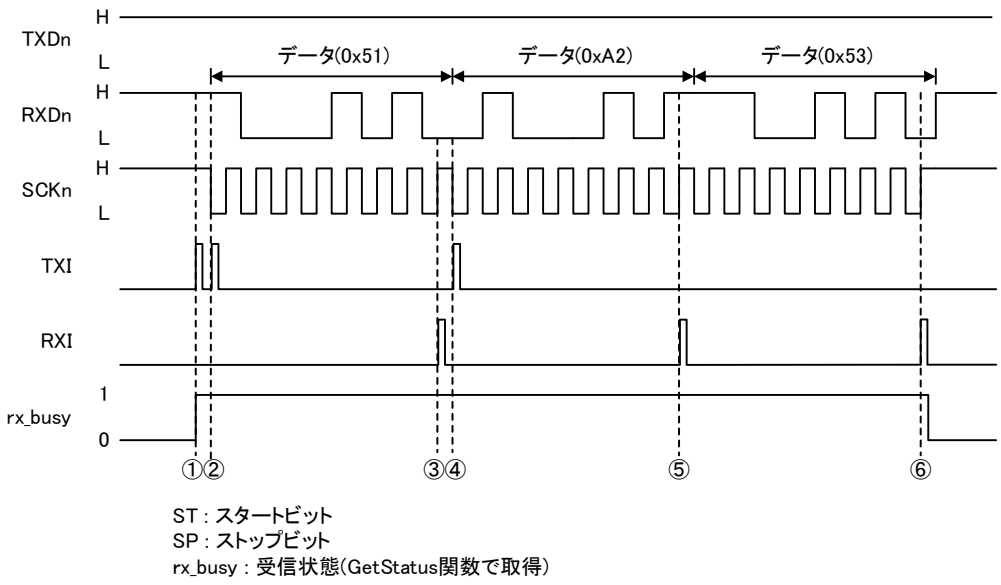


図 3-16 割り込み制御による受信動作（3 バイト受信、送信許可状態、ダミーデータ 0xFF）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)になります。また、TXI 割り込みが発生し、ダミーデータを送信データレジスタ（TDR）に書き込みます。

② 2 回目の TXI 割り込みにて 2 バイト目のダミーデータを TDR レジスタに書き込みます。

③ 1 バイト受信が完了すると、RXI 割り込みが発生し、受信データレジスタ（RDR）の値を指定されたバッファに読み出します。

④ 指定バイト数書き込み時の TXI 割り込みにて、TXI 割り込みを禁止にします。

⑤ 1 バイト受信完了ごとに RXI 割り込みが発生し、RDR レジスタから受信データを読み出します。

⑥ 最終データ読み出し時の RXI 割り込みで、rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注　受信エラーが発生した場合、ERI 割り込みが発生し、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

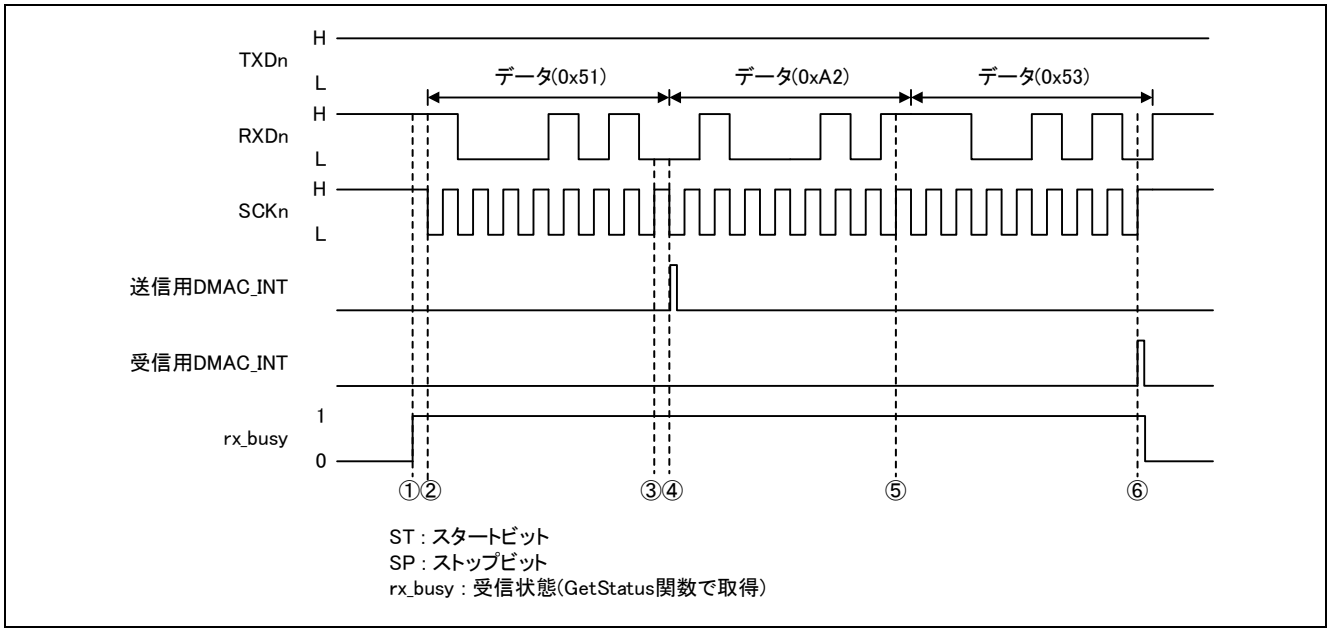


図 3-17 DMAC 制御による受信動作（3 バイト受信、送信許可状態、ダミーデータ 0xFF）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)に設定、DMAC の転送要因に TXI 割り込み、RXI 割り込みを設定し、最初のダミーデータを送信データレジスタ（TDR）に書き込みます。

② DMA 転送にて 2 バイト目以降のダミーデータが送信データレジスタ（TDR）に転送されます。

③ RXDn 端子からデータを受信すると、DMA 転送にて受信データレジスタ（RDR）の値を指定されたバッファに転送します。

④ 指定バイト数書き込み完了時、送信側の DMAC 転送完了割り込みが発生します。

⑤ 1 バイト受信完了ごとに DMA 転送にて、RDR レジスタの値を指定されたバッファに転送します。

⑥ 指定サイズの転送が完了後、受信側の DMAC 転送完了割り込みが発生します。割り込み処理にて rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注　受信エラーが発生した場合、ERI 割り込みが発生し、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

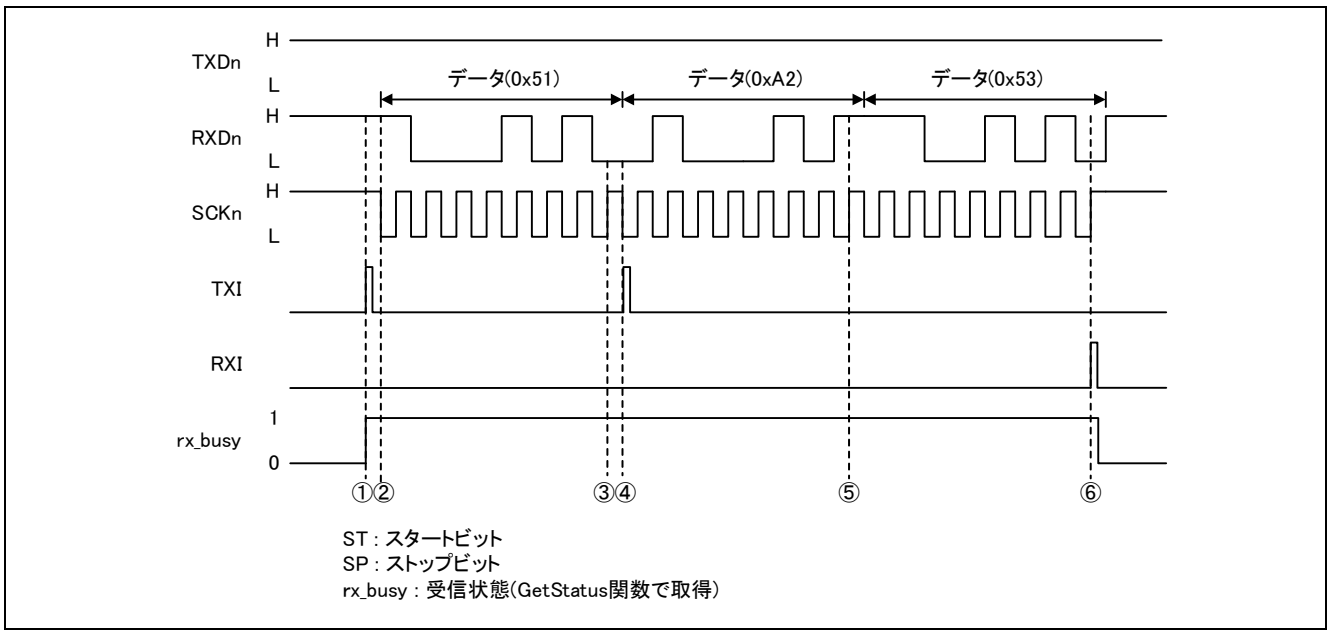


図 3-18 DTC 制御による受信動作（3 バイト受信、送信許可状態、ダミーデータ 0xFF）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)に設定、DTC 転送要因に TXI 割り込みを設定し、TXI 割り込みを許可にします。このとき、TXI 割り込みが発生し、最初のダミーデータを送信データレジスタ（TDR）に書き込みます。

② DMA 転送にて 2 バイト目以降のダミーデータが送信データレジスタ（TDR）に転送されます。

③ RXDn 端子からデータを受信すると、DMA 転送にて受信データレジスタ（RDR）の値を指定されたバッファに転送します。

④ 指定バイト数書き込み完了時、TXI 割り込みが発生します。

⑤ 1 バイト受信完了ごとに DMA 転送にて、RDR レジスタの値を指定されたバッファに転送します。

⑥ 最終データの転送完了後、RXI 割り込みが発生します。割り込み処理にて rx_busy フラグを”0”(受信待ち状態)にし、コールバック関数が登録されている場合、
ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注　受信エラーが発生した場合、ERI 割り込みが発生し、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

3.2.4 クロック同期マスタモードでの送受信処理

クロック同期マスタモードで送受信を行う手順を図 3-19 に示します。

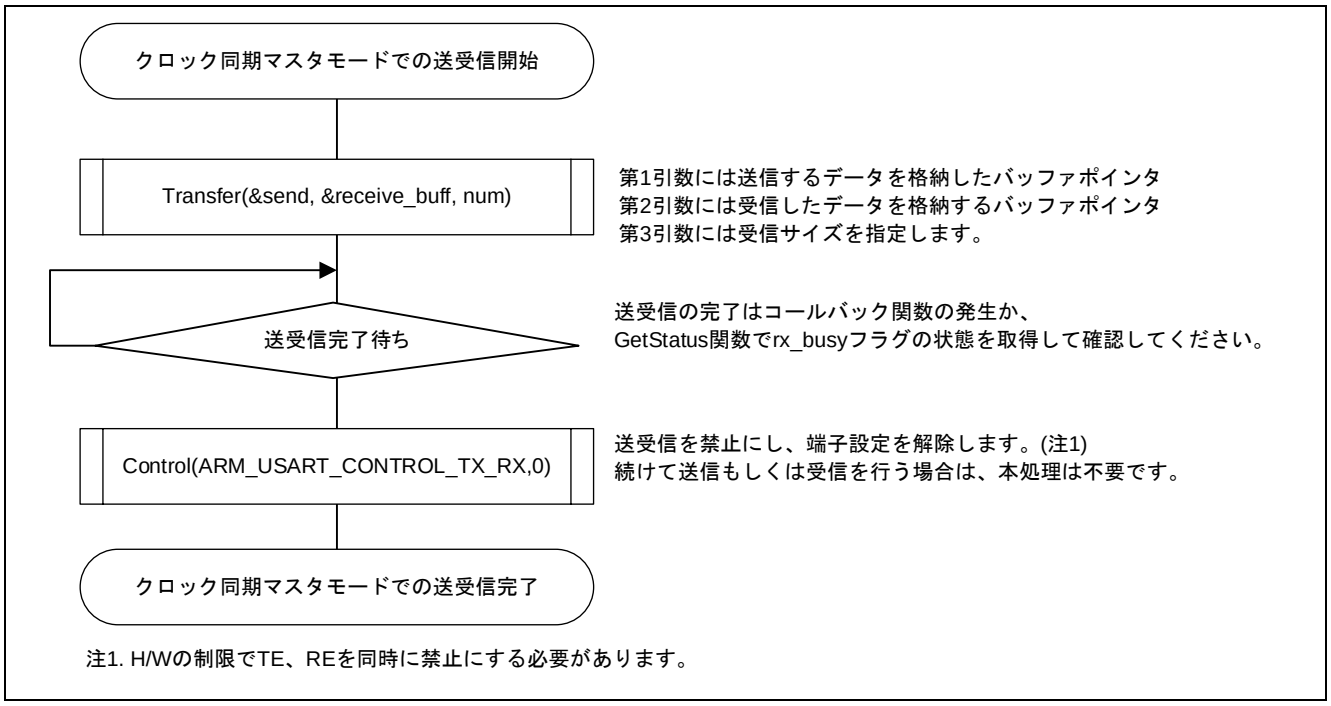


図 3-19 クロック同期マスタモードでの送受信手順

コールバック関数を設定していた場合、送受信が完了すると ARM_USART_EVENT_TRANSFER_COMPLETE を引数にコールバック関数が呼び出されます。

また、受信エラーが発生した場合、エラーイベント情報を引数にコールバック関数が呼び出され、送受信処理を完了します。クロック同期マスタモードの送受信処理で発生するエラーイベント情報を表 3-3 に示します。

表 3-3 クロック同期マスタモードの送受信処理で発生するエラーイベント情報

エラーイベント情報	内容
ARM_USART_EVENT_RX_OVERFLOW	オーバランエラー発生

クロック同期マスタモードによる送受信処理は、通信制御の設定が割り込み、または DMAC、または DTC にて処理方法が異なります。図 3-20 に通信制御が割り込みの場合の動作を、図 3-21 に通信制御が DMAC の場合の動作を、図 3-22 に通信制御が DTC の場合の動作を示します。

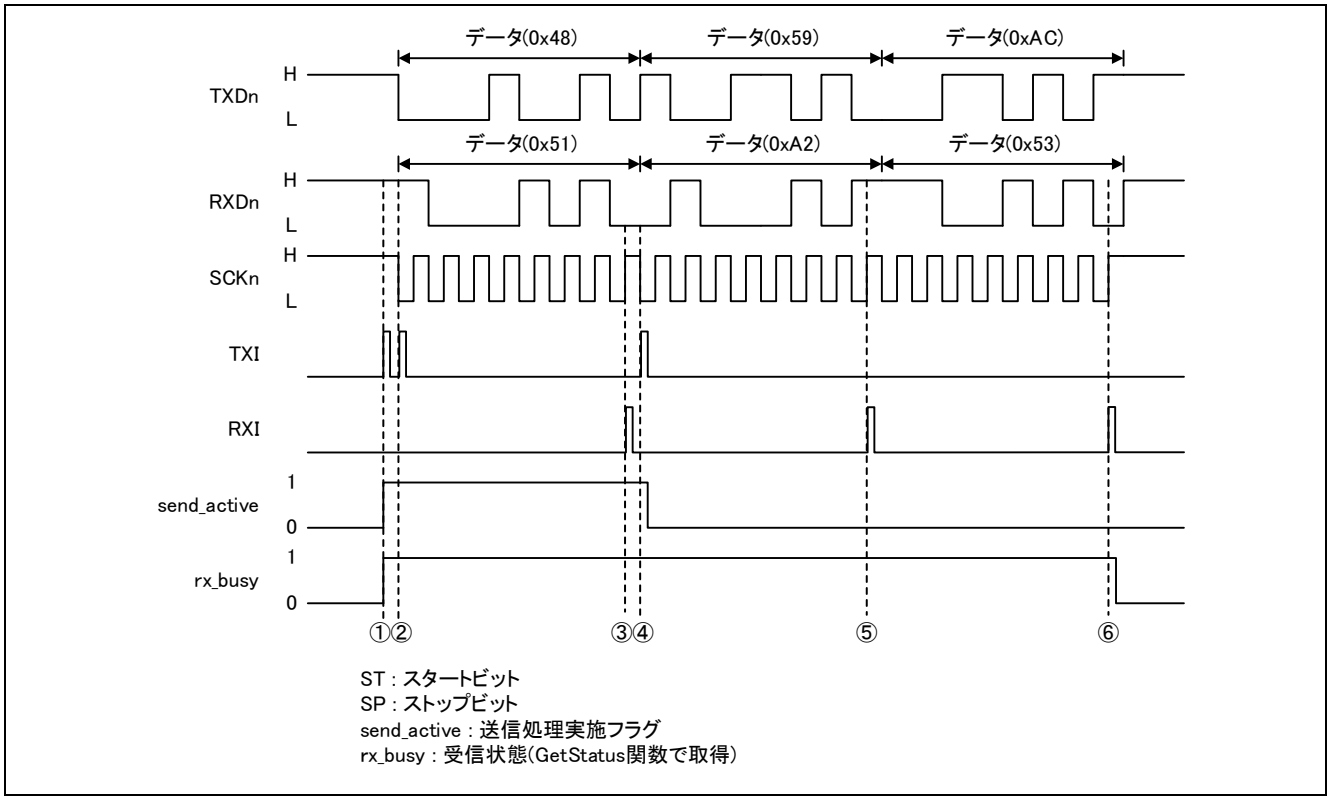


図 3-20 割り込み制御による送受信動作（3 バイト受信）

① Transfer 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に、rx_busy フラグが”1”(受信ビジー)に設定し、TXI 割り込みを許可にします。このとき、TXI 割り込みが発生し、1 バイト目の送信データを送信データレジスタ（TDR）に書き込みます。

② 2 回目の TXI 割り込みにて 2 バイト目の送信データを TDR レジスタに書き込みます。

③ 1 バイト受信が完了すると、RXI 割り込みが発生し、受信データレジスタ（RDR）の値を指定されたバッファに読み出します。

④ 指定バイト数書き込み時の TXI 割り込みにて、send_active フラグを”0”(送信レディ)に、TXI 割り込みを禁止にします。

⑤ 1 バイト受信完了ごとに RXI 割り込みが発生し、RDR レジスタから受信データを読み出します。

⑥ 最終データ読み出し時の RXI 割り込みで、rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注)

注 受信エラーが発生した場合、ERI 割り込みが発生し、tx_busy フラグを”0”(送信待ち状態)に、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

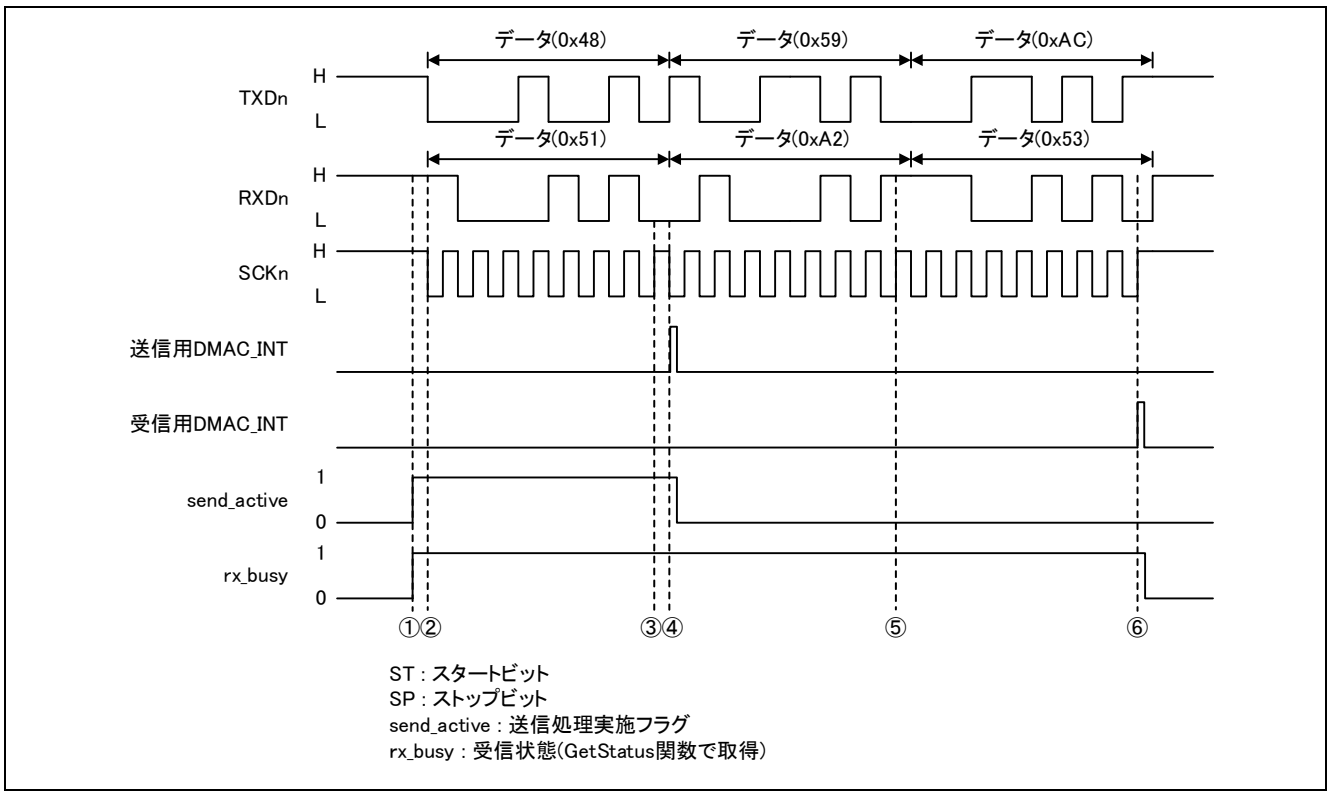


図 3-21 DMAC 制御による送受信動作（3 バイト受信）

① Transfer 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に、rx_busy フラグが”1”(受信ビジー)に設定、DMAC の転送要因に TXI 割り込み、RXI 割り込みを設定し、1 バイト目の送信データを送信データレジスタ（TDR）に書き込みます。

② DMA 転送にて 2 バイト目以降の送信データが送信データレジスタ（TDR）に転送されます。

③ RXDn 端子からデータを受信すると、DMA 転送にて受信データレジスタ（RDR）の値を指定されたバッファに転送します。

④ 指定バイト数書き込み完了時、送信側の DMAC 転送完了割り込みが発生します。割り込み処理にて send_active フラグを”0”(送信レディ)にします。

⑤ 1 バイト受信完了ごとに DMA 転送にて、RDR レジスタの値を指定されたバッファに転送します。

⑥ 指定サイズの転送が完了後、DMAC 転送完了割り込みが発生します。割り込み処理にて rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注)

注　受信エラーが発生した場合、ERI 割り込みが発生し、tx_busy フラグを”0”(送信待ち状態)に、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

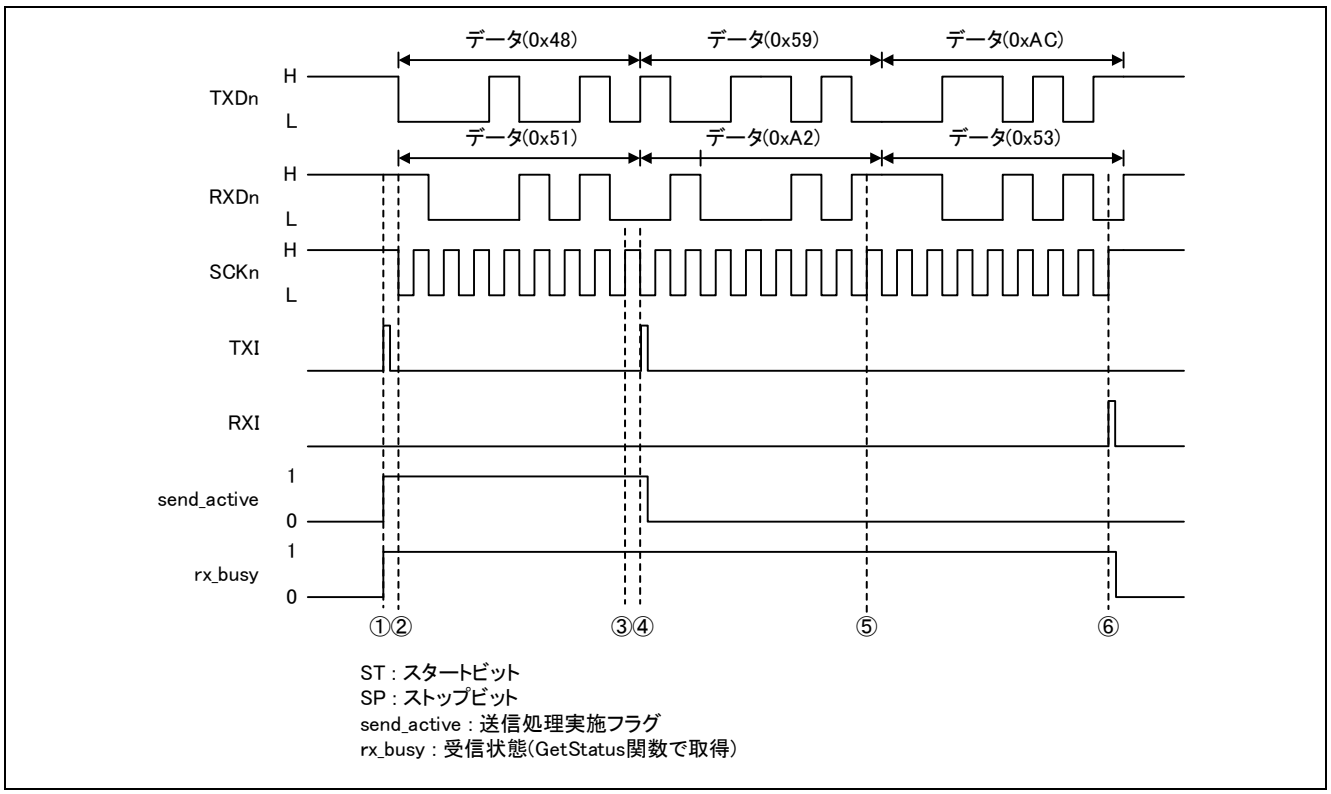


図 3-22 DTC 制御による送受信動作（3 バイト受信）

① Transfer 関数を実行すると、rx_busy フラグが”1”(受信ビジー)に設定、DTC 転送要因に TXI 割り込みを設定し、TXI 割り込みを許可にします。このとき、TXI 割り込みが発生し、1 バイト目の送信データを送信データレジスタ（TDR）に書き込みます。

② DMA 転送にて 2 バイト目以降の送信データが送信データレジスタ（TDR）に転送されます。

③ RXDn 端子からデータを受信すると、DMA 転送にて受信データレジスタ（RDR）の値を指定されたバッファに転送します。

④ 指定バイト数書き込み完了時、TXI 割り込みが発生します。割り込み処理にて send_active フラグを”0”(送信レディ)に、TXI 割り込みを禁止にします。

⑤ 1 バイト受信完了ごとに DMA 転送にて、RDR レジスタの値を指定されたバッファに転送します。

⑥ 最終データの転送完了後、RXI 割り込みが発生します。割り込み処理にて rx_busy フラグを”0”(受信待ち状態)にし、コールバック関数が登録されている場合、
ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注 受信エラーが発生した場合、ERI 割り込みが発生し、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

3.3 クロック同期スレーブモード

3.3.1 クロック同期スレーブモード初期設定手順

クロック同期スレーブモードの初期設定手順を図 3-23 に示します。

送信・受信を許可にする場合、r_system_cfg.h にて使用する割り込みを NVIC に登録してください。詳細は「2.4 通信制御」を参照してください。

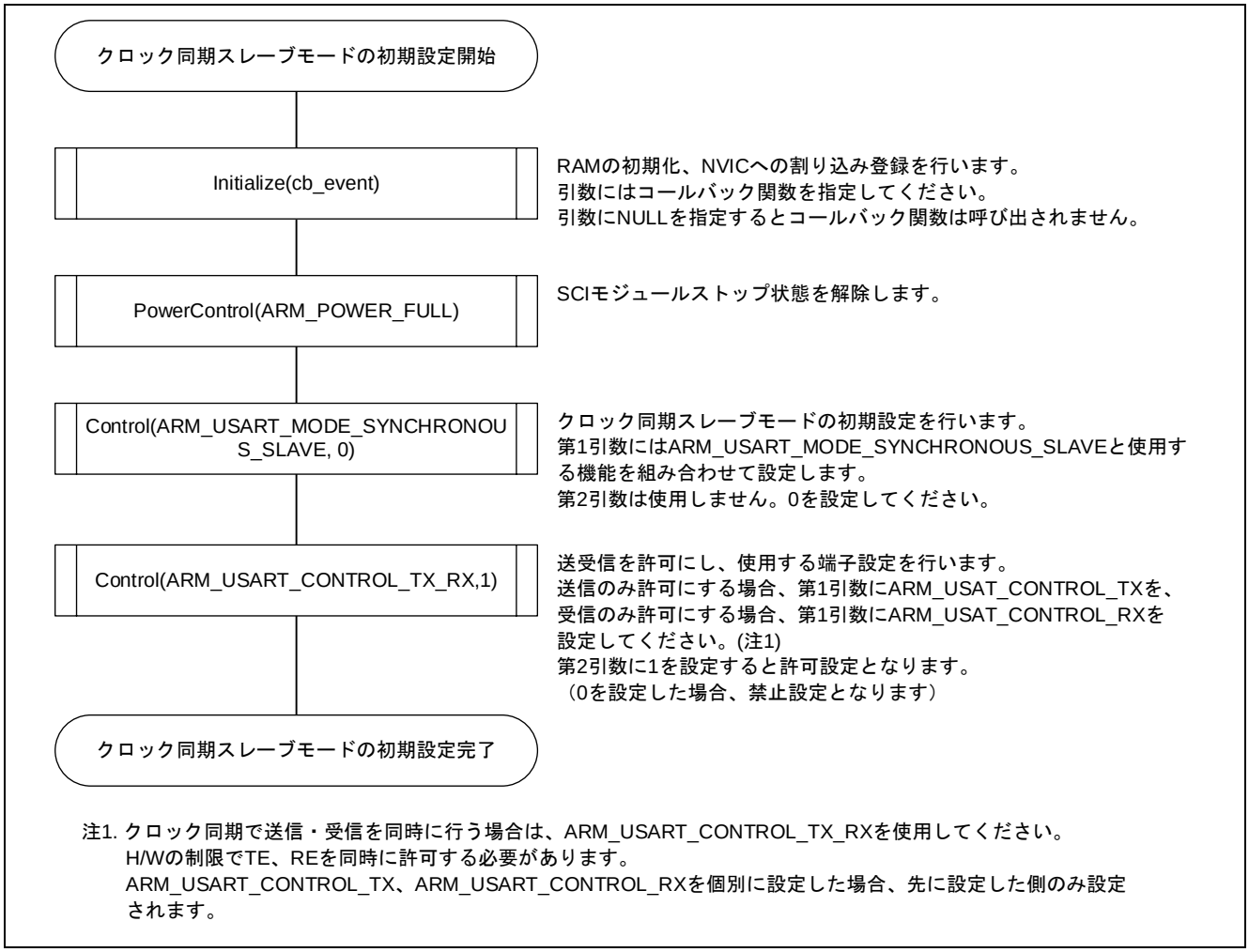


図 3-23 クロック同期スレーブモードの初期化手順

3.3.2 クロック同期スレーブモードでの送信処理

クロック同期スレーブモードで送信を行う手順を図 3-24 に示します。

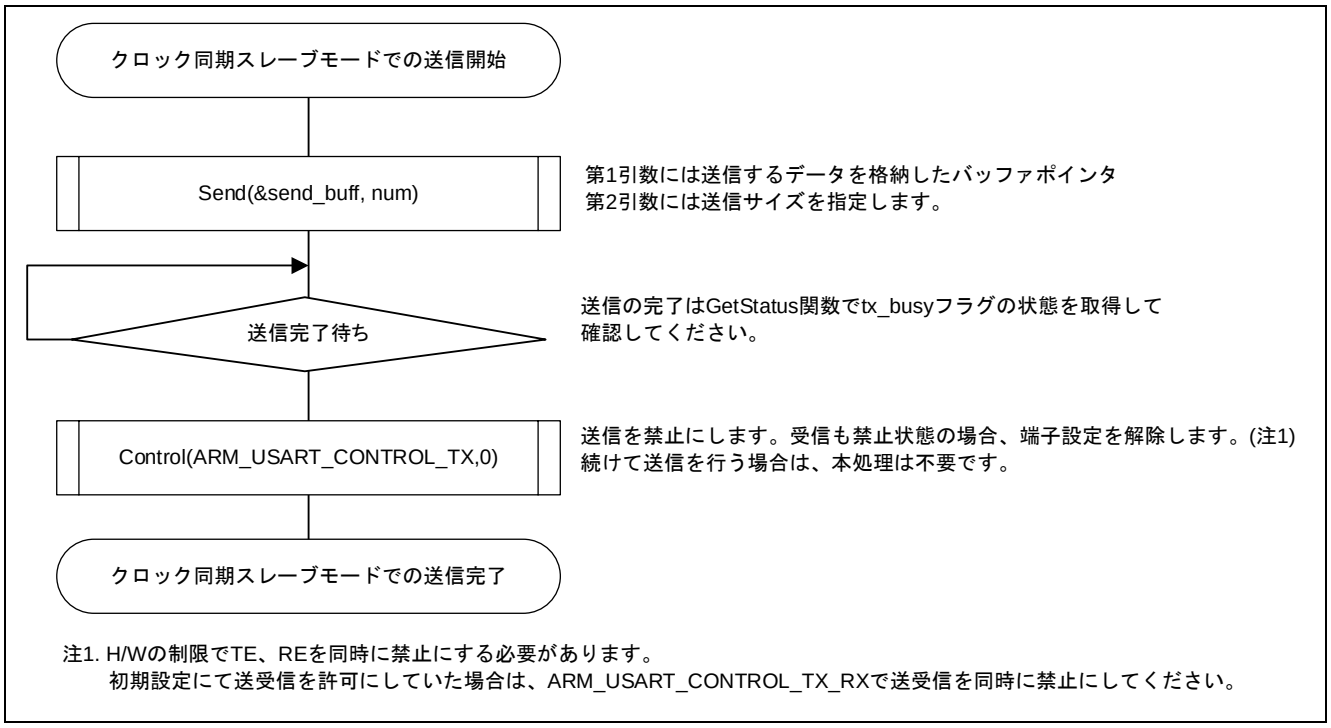


図 3-24 クロック同期スレーブモードでの送信手順

コールバック関数を設定していた場合、送信が完了すると ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数が呼び出されます。

クロック同期スレーブモードによる送信処理は、通信制御の設定が割り込み、または DMAC、または DTC にて処理方法が異なります。図 3-25 に通信制御が割り込みの場合の動作を、図 3-26 に通信制御が DMAC の場合の動作を、図 3-27 に通信制御が DTC の場合の動作を示します。

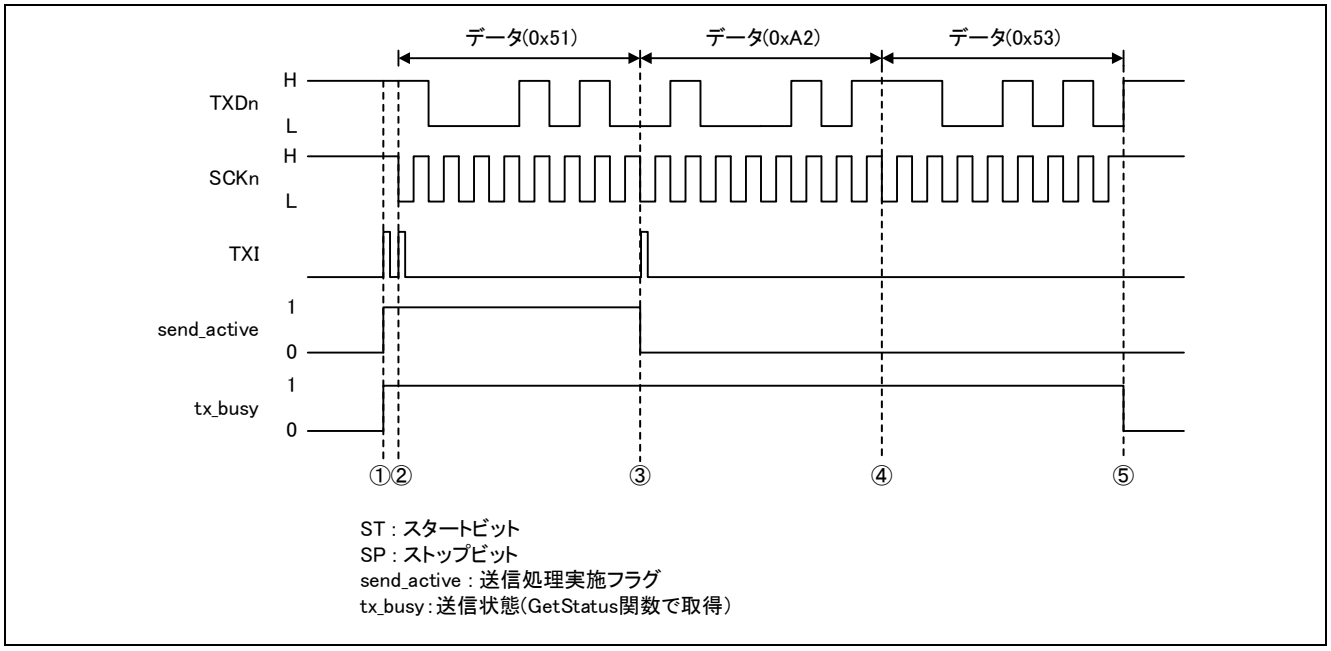


図 3-25 割り込み制御による送信動作（3 バイト送信）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)になります。また、TXI 割り込みが発生し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② SCKn 端子にクロックが入力されると 1 バイト目のデータが TXD 端子から出力を開始、2 回目の TXI 割り込みが発生します。割り込み処理にて 2 バイト目の送信データを TDR レジスタに書き込みます。

③ 最終データ書き込み後の TXI 割り込みで、TXI 割り込みを禁止にし、send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 2 バイト目のデータが出力されたのち、③で書き込んだ最終データが出力されます。(注)

⑤ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

注 tx_busy フラグが”0”の状態、かつ送信バッファに値がある状態（③～④の期間）で Send 関数を実行した場合、tx_busy フラグが”1”にして、Sned 関数を終了します。送信バッファが空になった時点（④のタイミング）で TXI 割り込みが発生し、1 バイト目のデータを TDR レジスタに書き込みます。

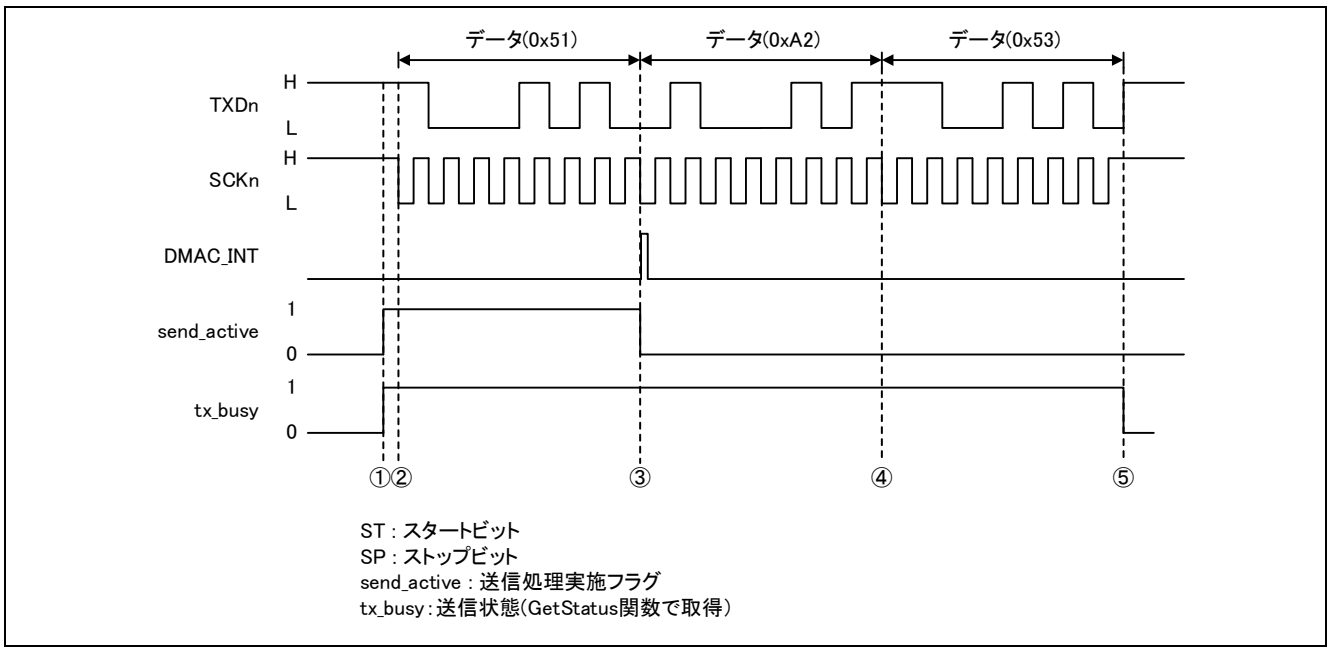


図 3-26 DMAC 制御による送信動作（3 バイト送信）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に設定、DMAC の転送要因に TXI 割り込みを設定し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② SCKn 端子にクロックが入力されると DMA 転送にて 2 バイト目以降のデータが送信データレジスタ（TDR）に転送されます。

③ DMA 転送にて最終データが転送されると、DMAC 転送完了割り込みが発生し、send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 2 バイト目のデータが出力されたのち、③で書き込んだ最終データが出力されます。(注)

⑤ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

注 tx_busy フラグが”0”の状態、かつ送信バッファに値がある状態（③～④の期間）で Send 関数を実行した場合、送信バッファが空になるまで（④のタイミング）待ったのち、①の処理を実行します。
待ち時間は r_usart_cfg.h の SCI_CHECK_TDRE_TIMEOUT 定義の値で変更できます。指定した時間、送信バッファが空にならなかった場合、Send 関数は ARM_DRIVER_ERROR_BUSY を返します。

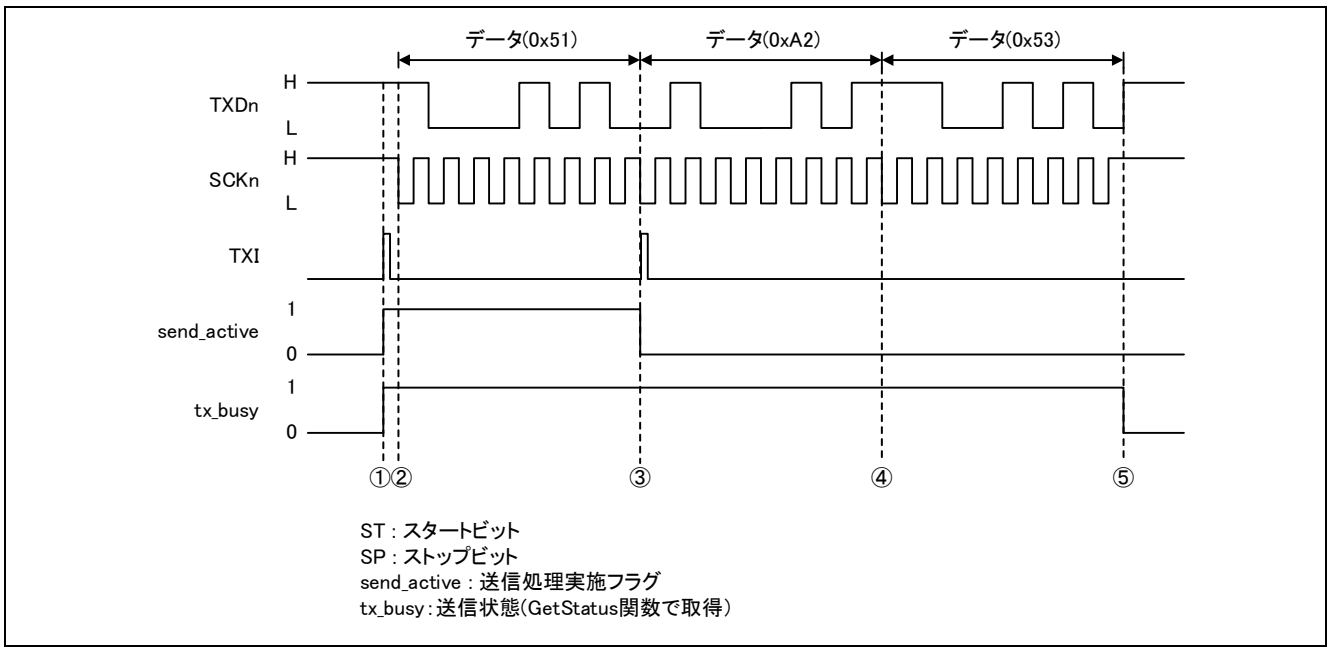


図 3-27 DTC 制御による送信動作（3 バイト送信）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に設定、DTC 転送要因に TXI 割り込みを設定し、TXI 割り込みを許可にします。このとき、TXI 割り込みが発生し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② SCKn 端子にクロックが入力されると DMA 転送にて 2 バイト目以降のデータが送信データレジスタ（TDR）に転送されます。

③ DMA 転送にて最終データが転送されると、TXI 割り込みが発生し、send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 2 バイト目のデータが出力されたのち、③で書き込んだ最終データが出力されます。(注)

⑤ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

注 tx_busy フラグが”0”の状態、かつ送信バッファに値がある状態（③～④の期間）で Send 関数を実行した場合、送信バッファが空になるまで（④のタイミング）待ったのち、①の処理を実行します。
待ち時間は r_usart_cfg.h の SCI_CHECK_TDRE_TIMEOUT 定義の値で変更できます。指定した時間、送信バッファが空にならなかった場合、Send 関数は ARM_DRIVER_ERROR_BUSY を返します。

3.3.3 クロック同期スレーブモードでの受信処理

クロック同期スレーブモードで受信を行う手順を図 3-28 に示します。

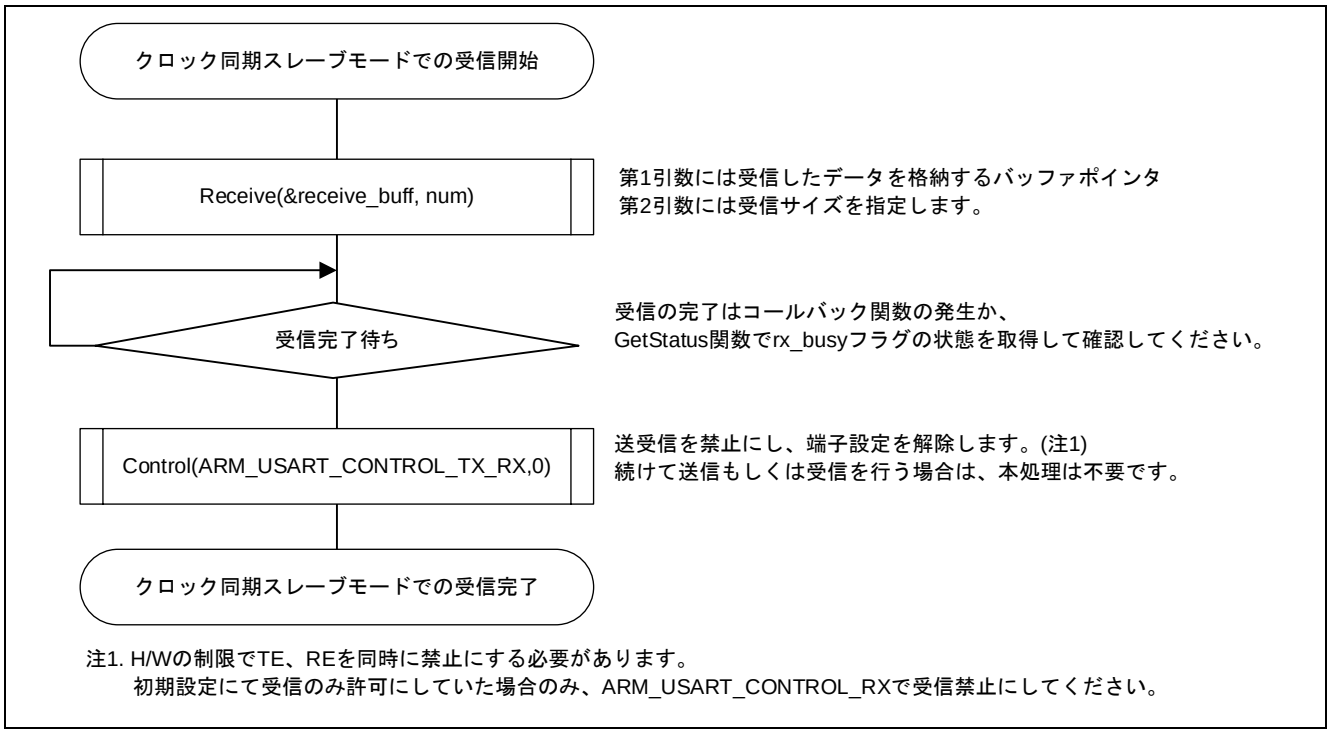


図 3-28 クロック同期スレーブモードでの受信手順

コールバック関数を設定していた場合、受信が完了すると ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数が呼び出されます。

また、受信エラーが発生した場合、エラーイベント情報を引数にコールバック関数が呼び出され、受信処理を完了します。クロック同期スレーブモードの受信処理で発生するエラーイベント情報を表 3-4 に示します。

表 3-4 クロック同期スレーブモードの受信処理で発生するエラーイベント情報

エラーイベント情報	内容
ARM_USART_EVENT_RX_OVERFLOW	オーバランエラー発生

クロック同期スレーブモードによる受信処理は、通信制御の設定が割り込み、または DMAC、または DTC にて処理方法が異なります。また、送信許可状態の場合、ダミーデータを送信バッファに書き込みます。出力するダミーデータは ARM_USART_SET_DEFAULT_TX_VALUE コマンドにて変更できます。

図 3-29 に通信制御が割り込みの場合の動作を、図 3-30 に通信制御が DMAC の場合の動作を、図 3-31 に通信制御が DTC の場合の動作を示します。

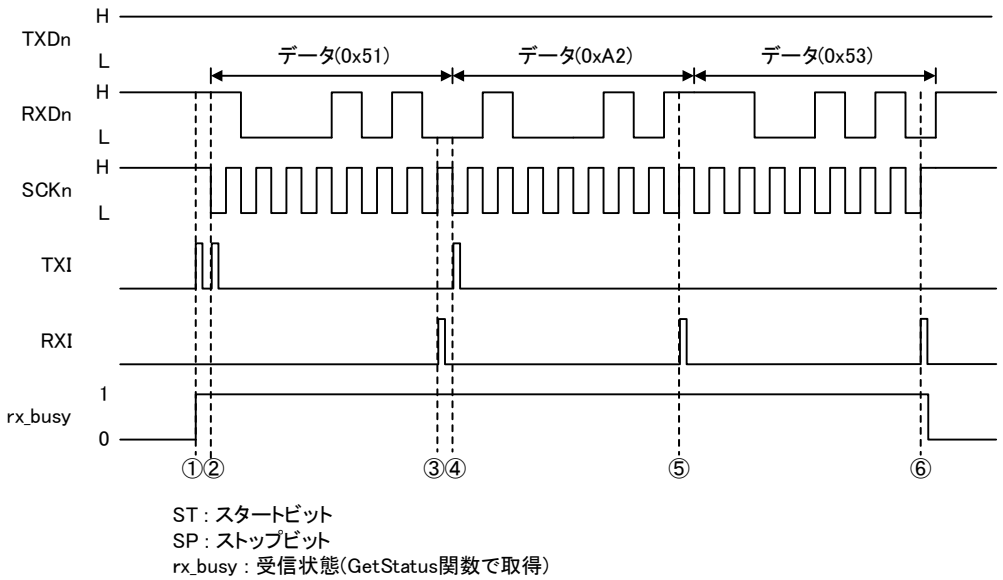


図 3-29 割り込み制御による受信動作（3 バイト受信、送信許可状態、ダミーデータ 0xFF）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)になります。また、TXI 割り込みが発生し、ダミーデータを送信データレジスタ（TDR）に書き込みます。

② SCKn 端子にクロックが入力されると、ダミーデータが TXD 端子から出力を開始します。2 回目の TXI 割り込みが発生し、2 バイト目のダミーデータを TDR レジスタに書き込みます。

③ 1 バイト受信が完了すると、RXI 割り込みが発生し、受信データレジスタ（RDR）の値を指定されたバッファに読み出します。

④ 指定バイト数書き込み時の TXI 割り込みにて、TXI 割り込みを禁止にします。

⑤ 1 バイト受信完了ごとに RXI 割り込みが発生し、RDR レジスタから受信データを読み出します。

⑥ 最終データ読み出し時の RXI 割り込みで、rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注　受信エラーが発生した場合、ERI 割り込みが発生し、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

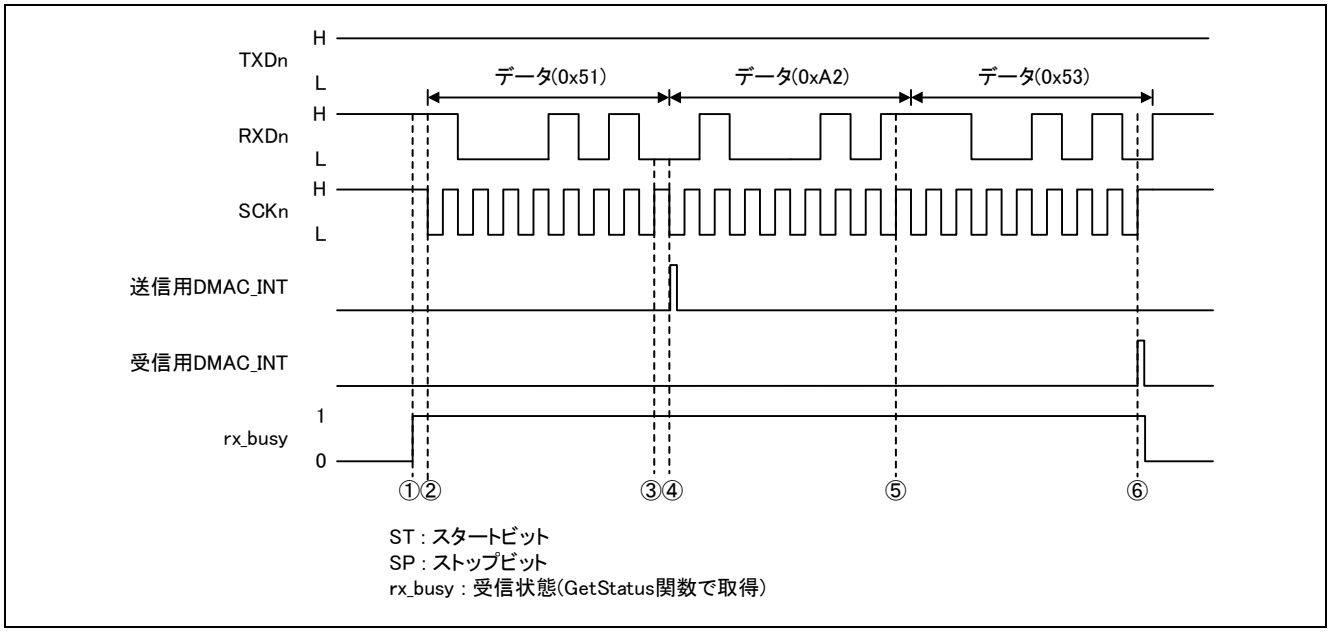


図 3-30 DMAC 制御による受信動作（3 バイト受信、送信許可状態、ダミーデータ 0xFF）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)に設定、DMAC の転送要因に TXI 割り込み、RXI 割り込みを設定し、最初のダミーデータを送信データレジスタ（TDR）に書き込みます。

② SCKn 端子にクロックが入力されると、ダミーデータが TXD 端子から出力を開始、DMA 転送にて 2 バイト目以降のダミーデータが送信データレジスタ（TDR）に転送されます。

③ RXDn 端子からデータを受信すると、DMA 転送にて受信データレジスタ（RDR）の値を指定されたバッファに転送します。

④ 指定バイト数書き込み完了時、送信側の DMAC 転送完了割り込みが発生します。

⑤ 1 バイト受信完了ごとに DMA 転送にて、RDR レジスタの値を指定されたバッファに転送します。

⑥ 指定サイズの転送が完了後、DMAC 転送完了割り込みが発生します。割り込み処理にて rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注　受信エラーが発生した場合、ERI 割り込みが発生し、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

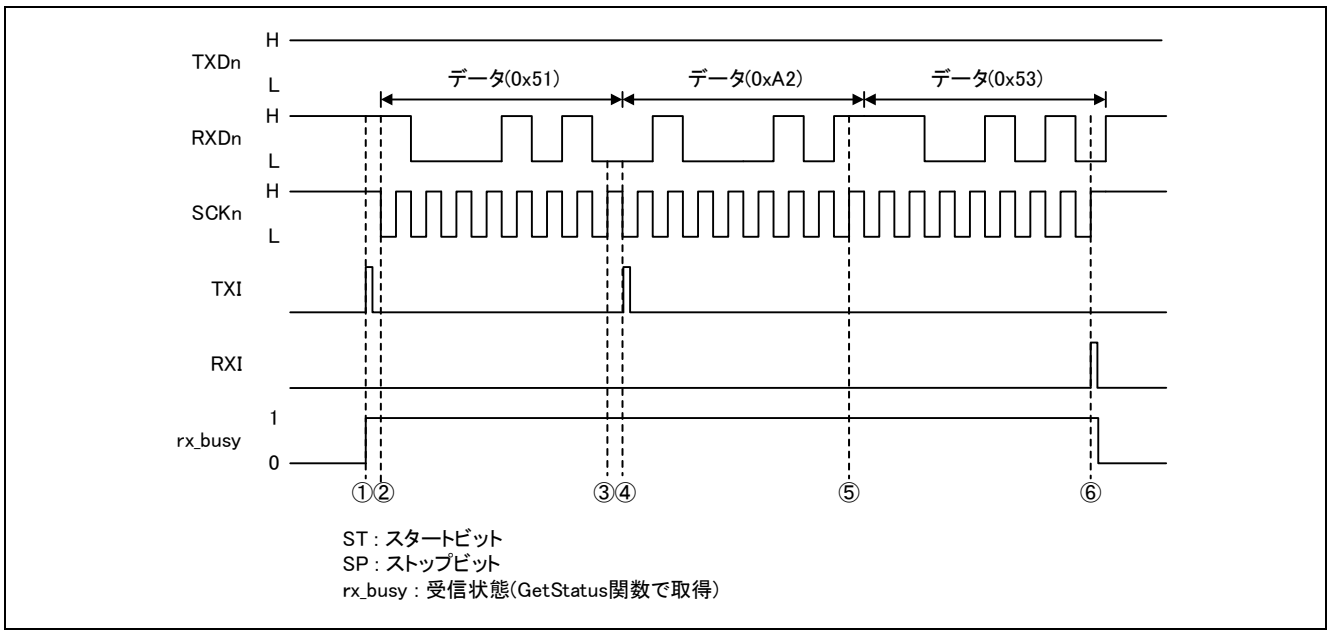


図 3-31 DTC 制御による受信動作（3 バイト受信、送信許可状態、ダミーデータ 0xFF）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)に設定、DTC 転送要因に TXI 割り込みを設定し、TXI 割り込みを許可にします。このとき、TXI 割り込みが発生し、最初のダミーデータを送信データレジスタ（TDR）に書き込みます。

② SCKn 端子にクロックが入力されると 1 バイト目のデータが TXD 端子から出力を開始､DMA 転送にて 2 バイト目以降のダミーデータが送信データレジスタ（TDR）に転送されます。

③ RXDn 端子からデータを受信すると、DMA 転送にて受信データレジスタ（RDR）の値を指定されたバッファに転送します。

④ 指定バイト数書き込み完了時、TXI 割り込みが発生します。

⑤ 1 バイト受信完了ごとに DMA 転送にて、RDR レジスタの値を指定されたバッファに転送します。

⑥ 最終データの転送完了後、RXI 割り込みが発生します。割り込み処理にて rx_busy フラグを”0”(受信待ち状態)にし、コールバック関数が登録されている場合、
ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注 受信エラーが発生した場合、ERI 割り込みが発生し、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

3.3.4 クロック同期スレーブモードでの送受信処理

クロック同期スレーブモードで送受信を行う手順を図 3-32 に示します。

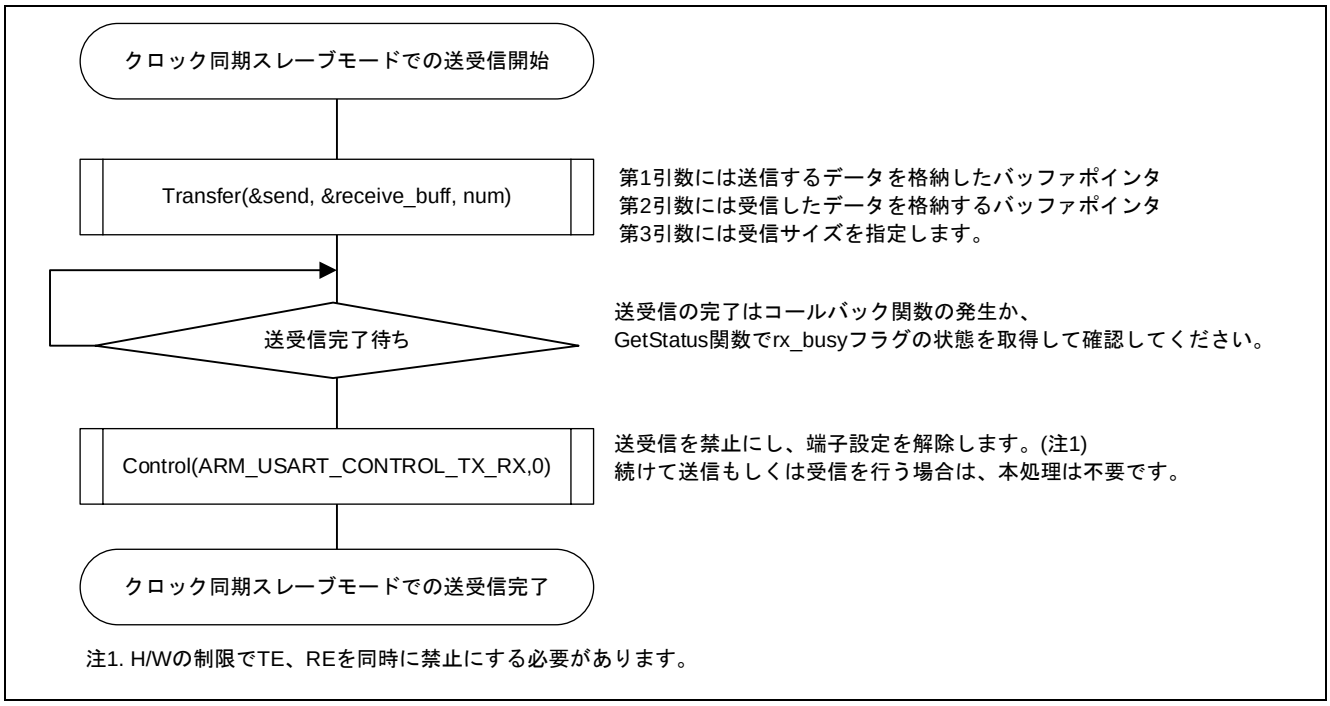


図 3-32 クロック同期スレーブモードでの送受信手順

コールバック関数を設定していた場合、受信が完了すると ARM_USART_EVENT_TRANSFER_COMPLETE を引数にコールバック関数が呼び出されます。

また、受信エラーが発生した場合、エラーイベント情報を引数にコールバック関数が呼び出され、送受信処理を完了します。クロック同期スレーブモードの送受信処理で発生するエラーイベント情報を表 3-5 に示します。

表 3-5 クロック同期スレーブモードの送受信処理で発生するエラーイベント情報

エラーイベント情報	内容
ARM_USART_EVENT_RX_OVERFLOW	オーバランエラー発生

クロック同期スレーブモードによる送受信処理は、通信制御の設定が割り込み、または DMAC、または DTC にて処理方法が異なります。図 3-33 に通信制御が割り込みの場合の動作を、図 3-34 に通信制御が DMAC の場合の動作を、図 3-35 に通信制御が DTC の場合の動作を示します。

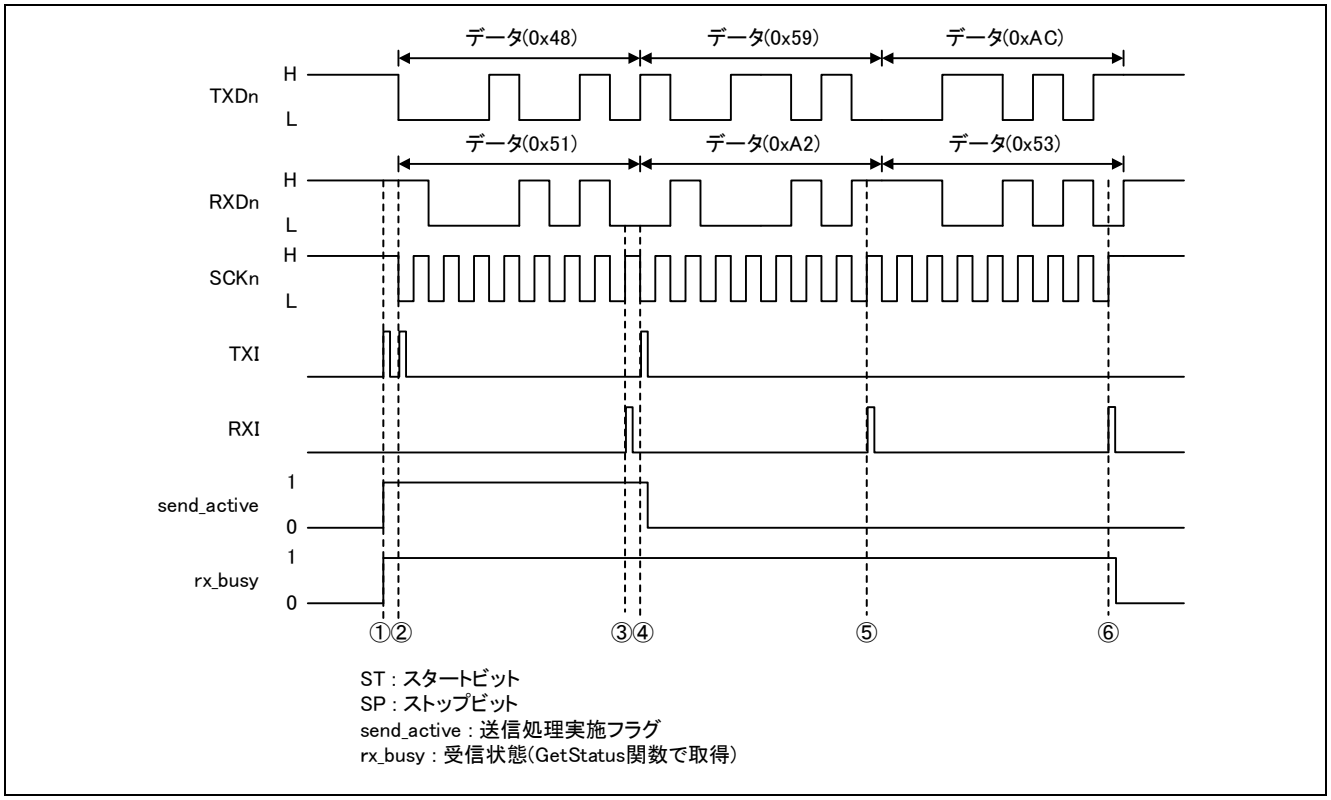


図 3-33 割り込み制御による送受信動作（3 バイト受信）

① Transfer 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に、rx_busy フラグが”1”(受信ビジー)に設定し、TXI 割り込みを許可にします。このとき、TXI 割り込みが発生し、1 バイト目の送信データを送信データレジスタ（TDR）に書き込みます。

② 2 回目の TXI 割り込みにて 2 バイト目の送信データを TDR レジスタに書き込みます。

③ 1 バイト受信が完了すると、RXI 割り込みが発生し、受信データレジスタ（RDR）の値を指定されたバッファに読み出します。

④ 指定バイト数書き込み時の TXI 割り込みにて、send_active フラグを”0”(送信レディ)に、TXI 割り込みを禁止にします。

⑤ 1 バイト受信完了ごとに RXI 割り込みが発生し、RDR レジスタから受信データを読み出します。

⑥ 最終データ読み出し時の RXI 割り込みで、rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注)

注　受信エラーが発生した場合、ERI 割り込みが発生し、tx_busy フラグを”0”(送信待ち状態)に、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

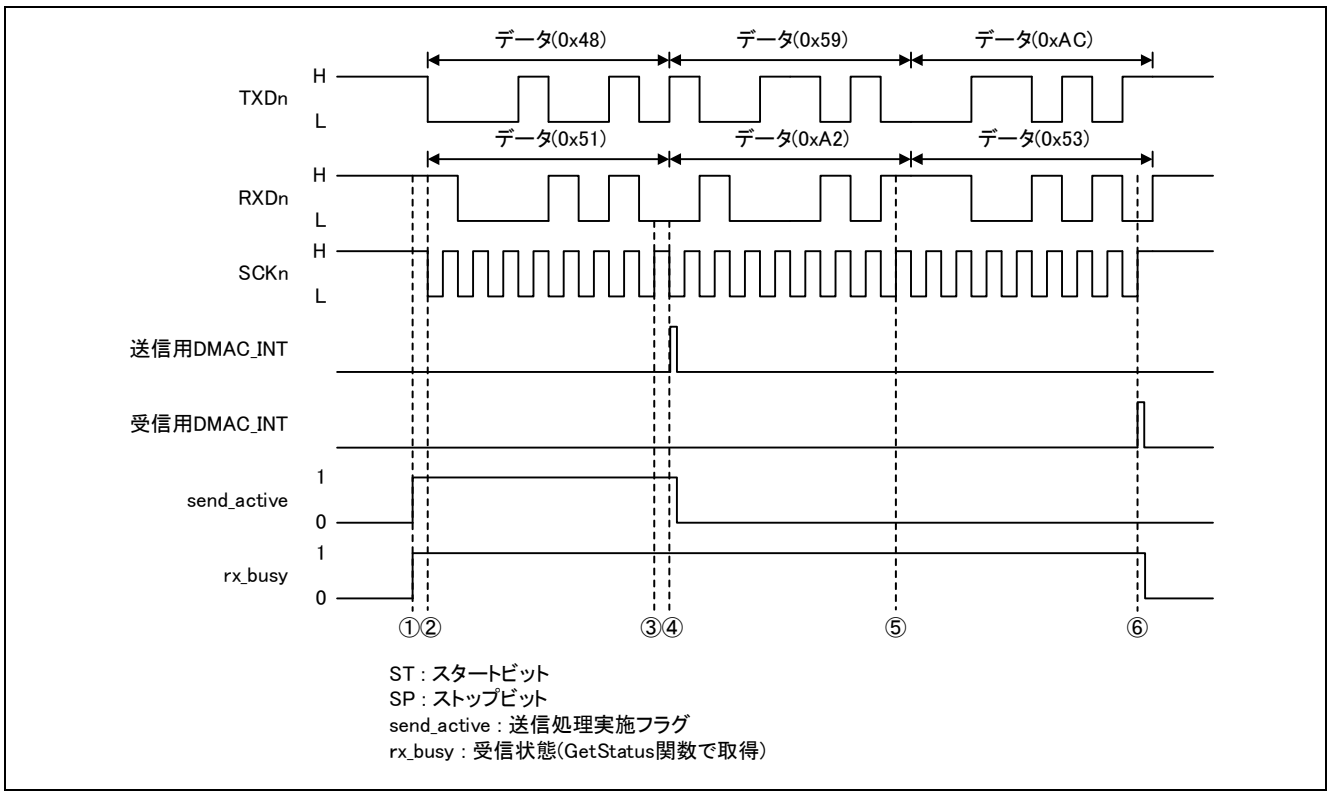


図 3-34 DMAC 制御による送受信動作（3 バイト受信）

① Transfer 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に、rx_busy フラグが”1”(受信ビジー)に設定、DMAC の転送要因に TXI 割り込み、RXI 割り込みを設定し、1 バイト目の送信データを送信データレジスタ（TDR）に書き込みます。

② DMA 転送にて 2 バイト目以降の送信データが送信データレジスタ（TDR）に転送されます。

③ RXDn 端子からデータを受信すると、DMA 転送にて受信データレジスタ（RDR）の値を指定されたバッファに転送します。

④ 指定バイト数書き込み完了時、送信側の DMAC 転送完了割り込みが発生します。割り込み処理にて send_active フラグを”0”(送信レディ)にします。

⑤ 1 バイト受信完了ごとに DMA 転送にて、RDR レジスタの値を指定されたバッファに転送します。

⑥ 指定サイズの転送が完了後、受信側の DMAC 転送完了割り込みが発生します。割り込み処理にて rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注)

注　受信エラーが発生した場合、ERI 割り込みが発生し、tx_busy フラグを”0”(送信待ち状態)に、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

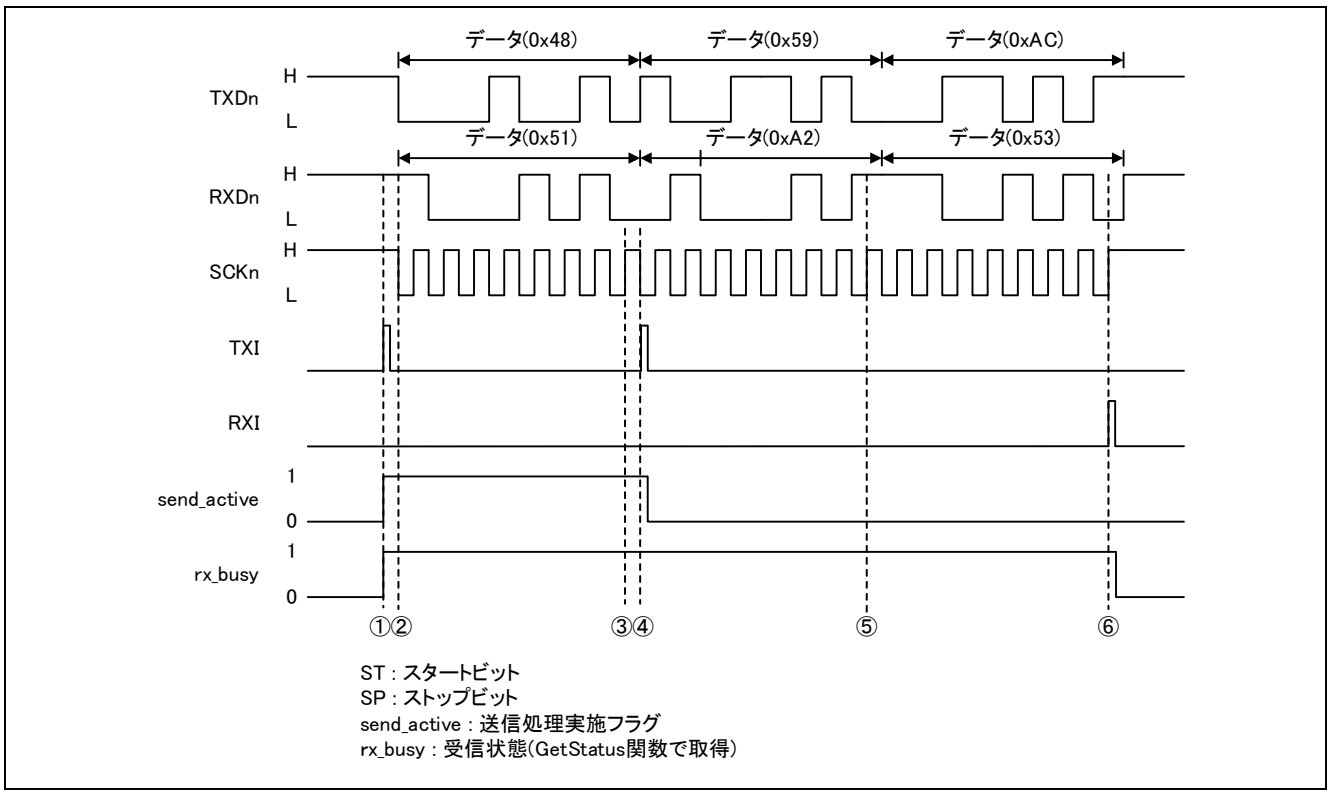


図 3-35 DTC 制御による受信動作（3 バイト受信）

① Transfer 関数を実行すると、rx_busy フラグが”1”(受信ビジー)に設定、DTC 転送要因に TXI 割り込みを設定し、TXI 割り込みを許可にします。このとき、TXI 割り込みが発生し、1 バイト目の送信データを送信データレジスタ（TDR）に書き込みます。

② DMA 転送にて 2 バイト目以降の送信データが送信データレジスタ（TDR）に転送されます。

③ RXDn 端子からデータを受信すると、DMA 転送にて受信データレジスタ（RDR）の値を指定されたバッファに転送します。

④ 指定バイト数書き込み完了時、TXI 割り込みが発生します。割り込み処理にて send_active フラグを”0”(送信レディ)に、TXI 割り込みを禁止にします。

⑤ 1 バイト受信完了ごとに DMA 転送にて、RDR レジスタの値を指定されたバッファに転送します。

⑥ 最終データの転送完了後、RXI 割り込みが発生します。割り込み処理にて rx_busy フラグを”0”(受信待ち状態)にし、コールバック関数が登録されている場合、
ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注 受信エラーが発生した場合、ERI 割り込みが発生し、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

3.4 スマートカードモード

3.4.1 スマートカードモード初期設定手順

スマートカードモードの初期設定手順を図 3-36 に示します。

送信・受信を許可にする場合、r_system_cfg.h にて使用する割り込みを NVIC に登録してください。詳細は「2.4 通信制御」を参照してください。

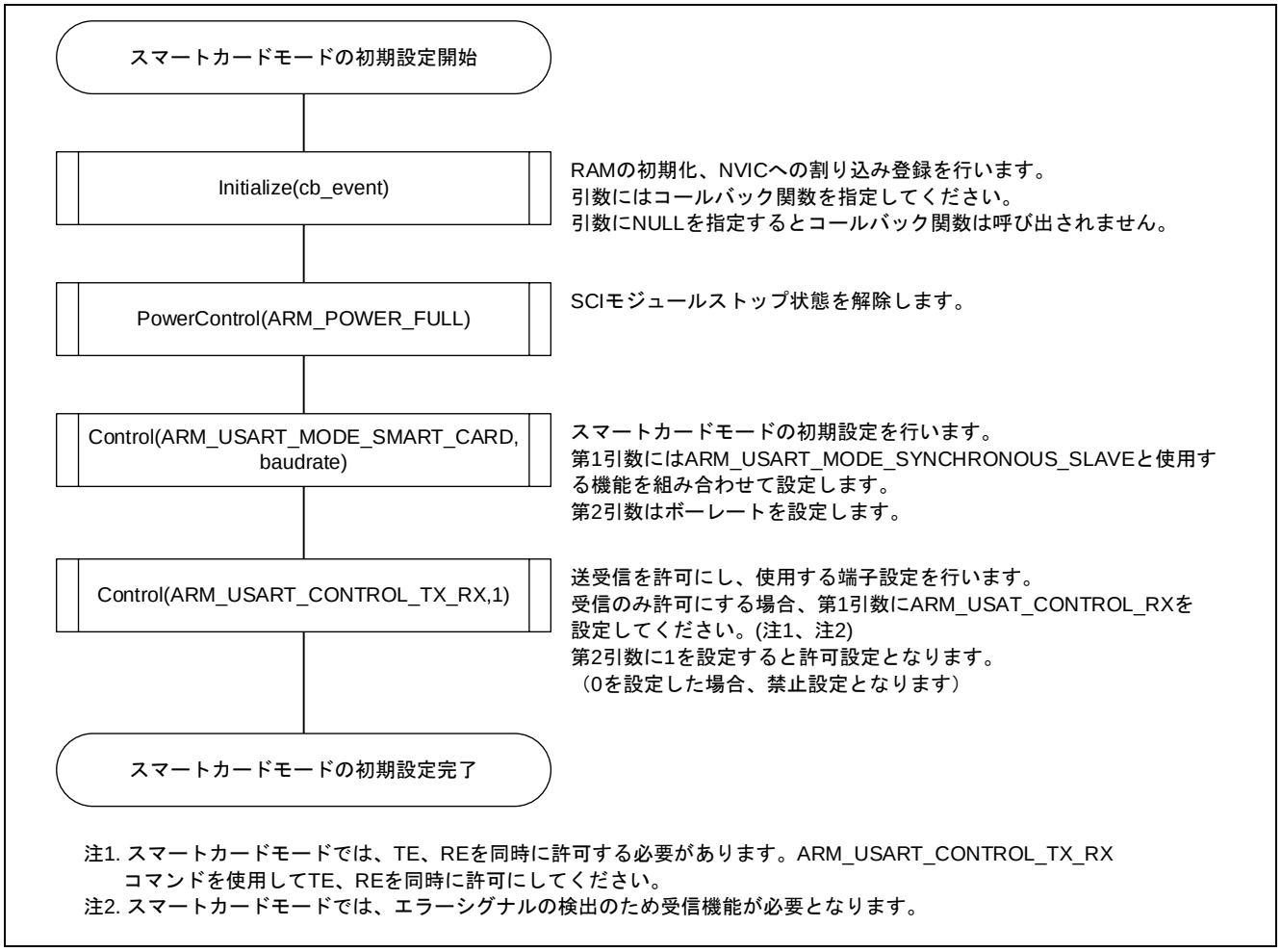


図 3-36 スマートカードモードの初期化手順

3.4.2 スマートカードモードでの送信処理

スマートカードモードで送信を行う手順を図 3-37 に示します。

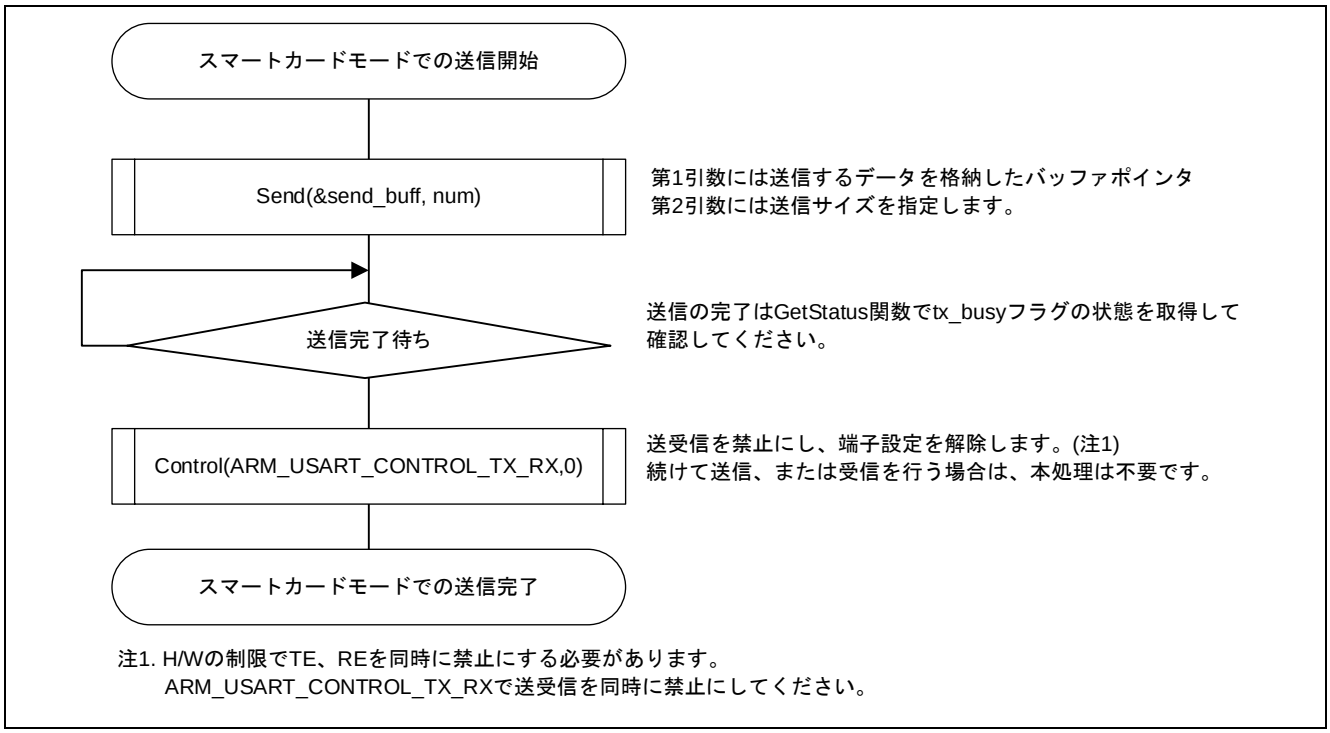


図 3-37 スマートカードモードでの送信手順

コールバック関数を設定していた場合、送信が完了すると ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数が呼び出されます。

スマートカードモードによる送信処理は、通信制御の設定が割り込み、または DMAC、または DTC にて処理方法が異なります。図 3-38 に通信制御が割り込みの場合の動作を、図 3-39 に通信制御が DMAC の場合の動作を、図 3-40 に通信制御が DTC の場合の動作を示します。

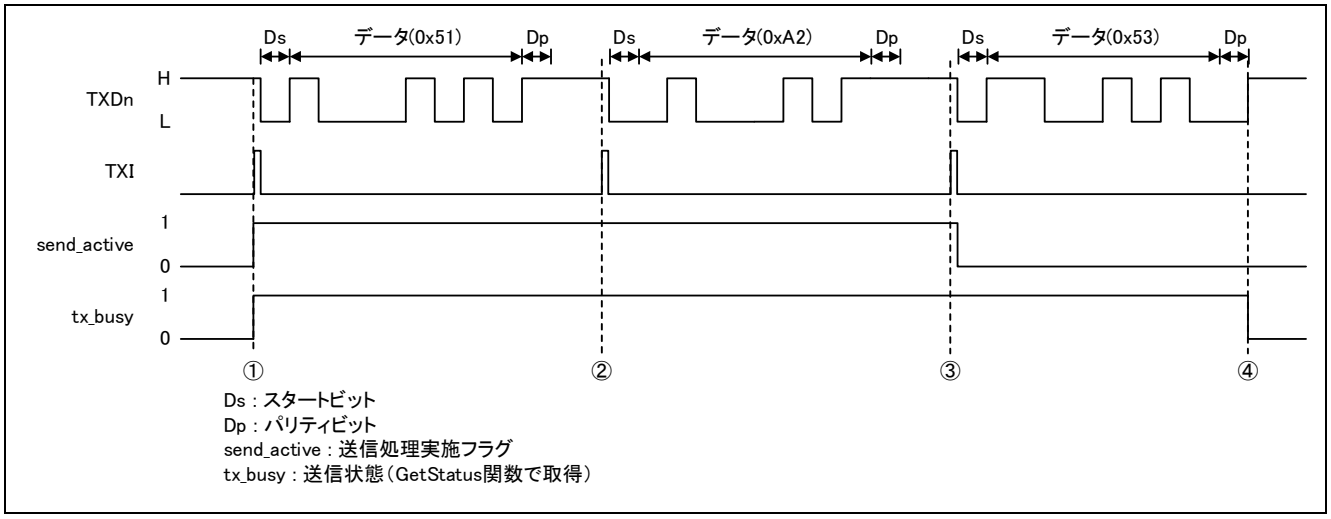


図 3-38 割り込み制御による送信動作（3 バイト送信、偶数パリティ）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)になります。また、TXI 割り込みが発生し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② 1 バイト目のデータが出力後、受信局からのエラーシグナルを受信しなかった場合、2 回目の TXI 割り込みが発生します。割り込み処理にて、2 バイト目の送信データを TDR レジスタに書き込みます。

③ 最終データ書き込み後の TXI 割り込みで、TXI 割り込みを禁止にし、send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

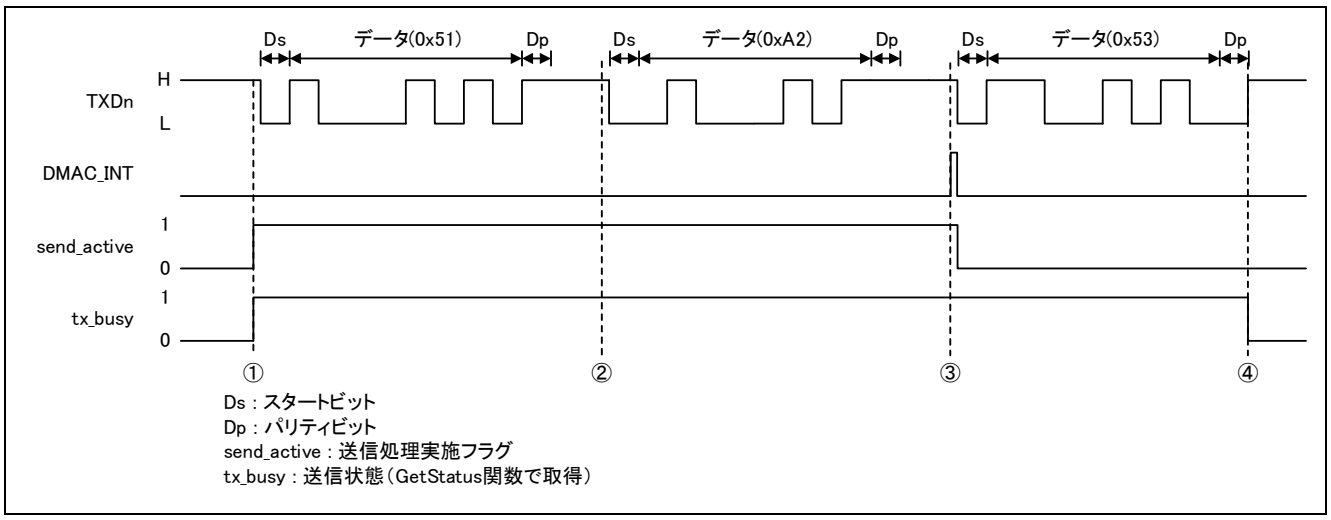


図 3-39 DMAC 制御による送信動作（3 バイト送信、偶数パリティ）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に設定、DMAC の転送要因に TXI 割り込みを設定し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② 1 バイト目のデータが出力後、受信局からのエラーシグナルを受信しなかった場合、DMA 転送にて 2 バイト目以降のデータが送信データレジスタ（TDR）に転送されます。

③ DMA 転送にて最終データが転送されると、DMAC 転送完了割り込みが発生し、send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

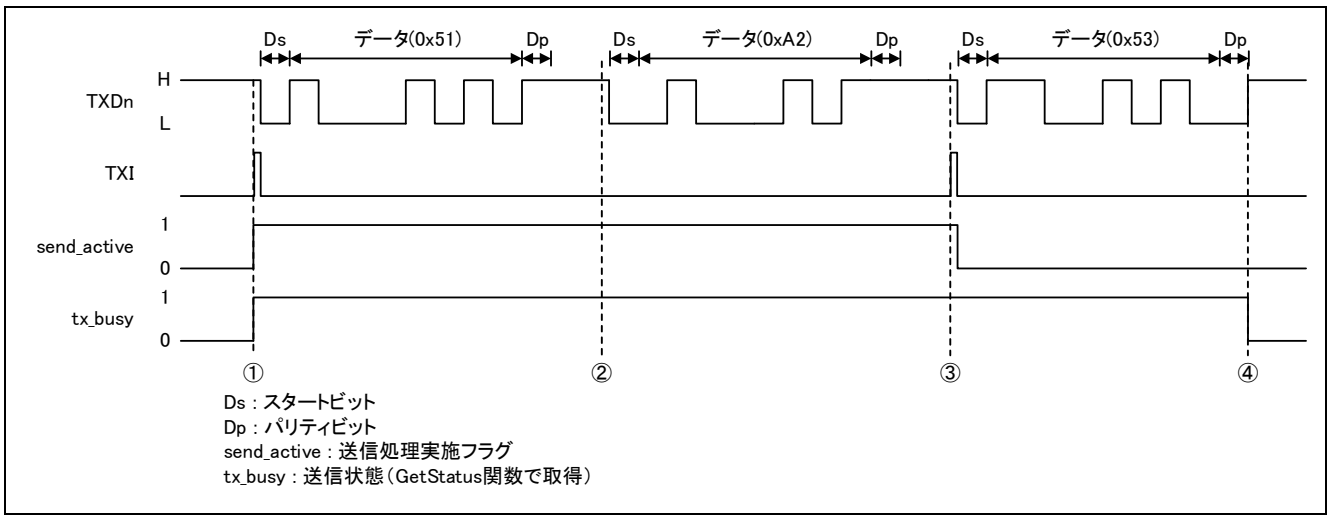


図 3-40 DTC 制御による送信動作（3 バイト送信、偶数パリティ）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)に設定、DTC 転送要因に TXI 割り込みを設定し、TXI 割り込みを許可にします。このとき、TXI 割り込みが発生し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② 1 バイト目のデータが出力後、受信局からのエラーシグナルを受信しなかった場合、DMA 転送にて 2 バイト目以降のデータが送信データレジスタ（TDR）に転送されます。

③ DMA 転送にて最終データが転送されると、TXI 割り込みが発生し、send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、
ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。

④ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

3.4.3 スマートカードモード送信時のエラーシグナル受信

スマートカードモードで送信中にエラーシグナルを受信すると、H/W にて自動的に同じデータを再送信します。また、ERI 割り込みが発生し、コールバック関数が登録されている場合は、ARM_USART_EVENT_RX_PARITY_ERROR を引数にコールバック関数が呼び出されます。

送信中にエラーシグナルを受信した場合の動作を図 3-42 に示します。

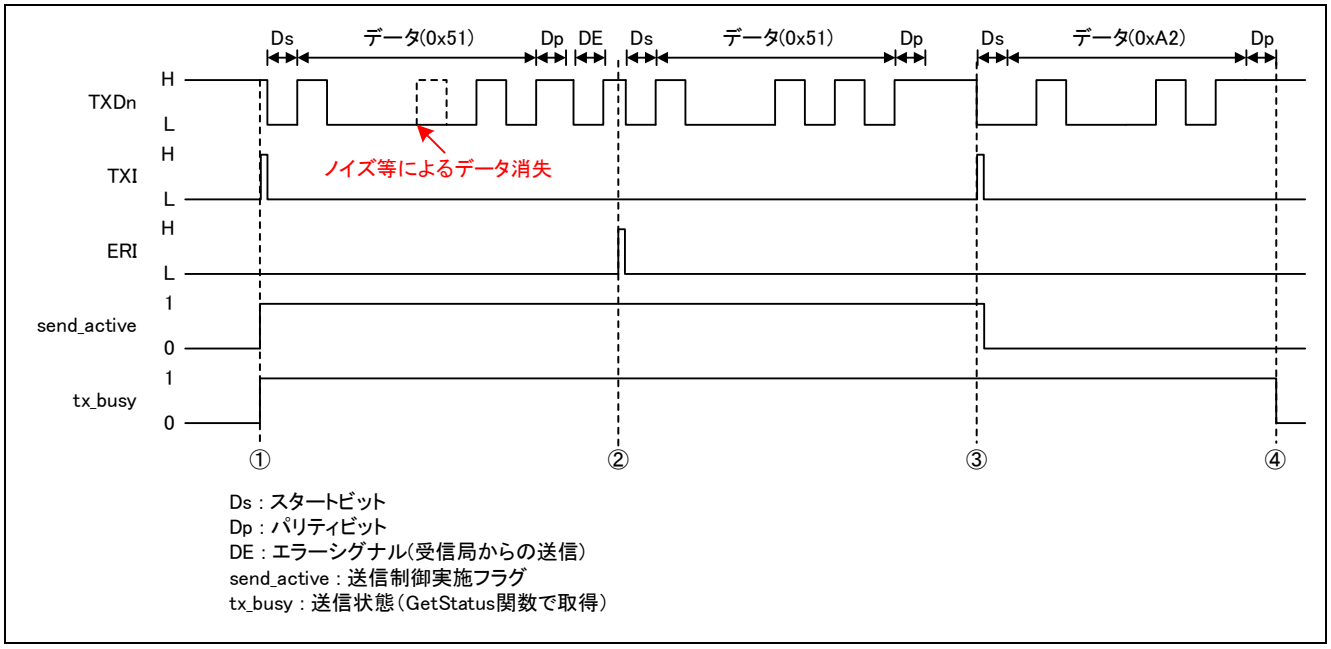


図 3-41 エラーシグナル受信時の動作（2 バイト送信、偶数パリティ）

① Send 関数を実行すると、tx_busy フラグが”1”(送信ビジー)になります。また、TXI 割り込みが発生し、1 バイト目のデータを送信データレジスタ（TDR）に書き込みます。

② 1 バイト目のデータが出力後、受信局からのエラーシグナルを受信した場合、ERI 割り込みが発生します。割り込み処理にて、コールバック関数が登録されている場合、ARM_USART_EVENT_RX_PARITY_ERROR を引数にコールバック関数を実行します。
また、自動的に 1 バイト目のデータが再送されます。

③ 再送したデータが出力後、受信局からのエラーシグナルを受信しなかった場合、2 回目の TXI 割り込みが発生します。割り込み処理にて、2 バイト目のデータ書き込み、TXI 割り込みを禁止設定、および send_active フラグを”0”(送信レディ)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_SEND_COMPLETE を引数にコールバック関数を実行します。(注)

④ 送信が完了すると tx_busy フラグが”0”(送信完了)になります。

注 最終データ出力後にエラーシグナルを受信した場合も②と同様、ERI 割り込みの発生およびデータの再送が行われます。

3.4.4 スマートカードモードでの受信処理

スマートカードモードで受信を行う手順を図 3-42 に示します。

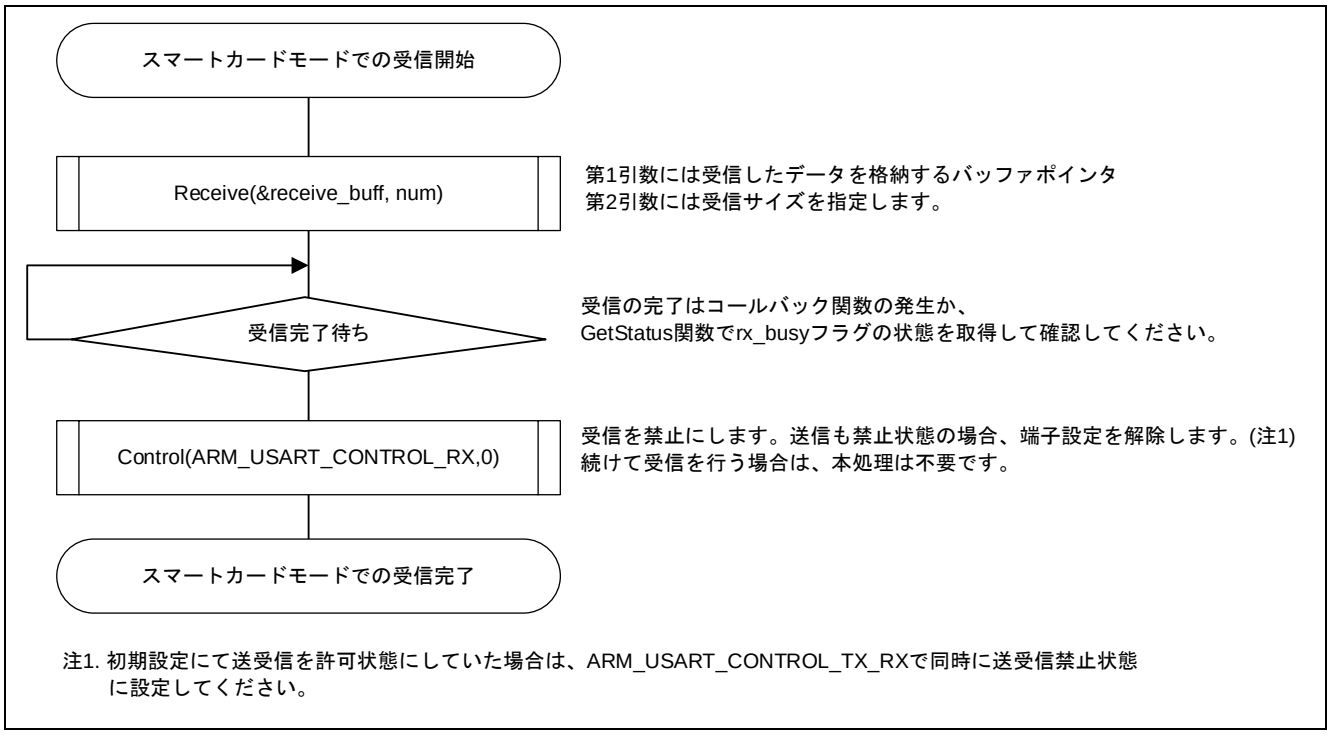


図 3-42 スマートカードモードでの受信手順

コールバック関数を設定していた場合、受信が完了すると ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数が呼び出されます。

また、受信エラーが発生した場合、エラーイベント情報を引数にコールバック関数が呼び出されます。受信エラーがオーバランエラーの場合、受信処理を完了します。パリティエラーの場合、エラーシグナルを出力し、受信制御を継続します。スマートカードモードの受信処理で発生するエラーイベント情報を表 3-6 に示します。

表 3-6 スマートカードモードの受信処理で発生するエラーイベント情報

エラーイベント情報	内容
ARM_USART_EVENT_RX_OVERFLOW	オーバランエラー発生
ARM_USART_EVENT_RX_PARITY_ERROR	パリティエラー発生

注1. 複数のエラーが発生した場合、OR 結合されたエラーイベント情報を引数にコールバック関数が呼び出されます。

スマートカードモードによる受信処理は、通信制御の設定が割り込み、または DMAC、または DTC にて処理方法が異なります。図 3-43 に通信制御が割り込みの場合の動作を、図 3-44 に通信制御が DMAC の場合の動作を、図 3-45 に通信制御が DTC の場合の動作を示します。

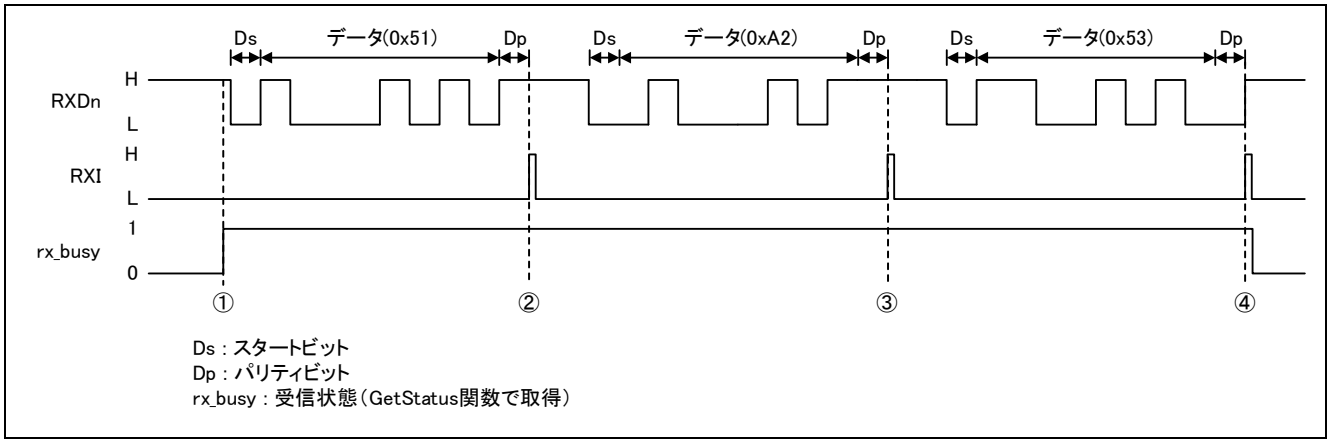


図 3-43 割り込み制御による受信動作（3 バイト受信、偶数パリティ）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)になります。

② 1 バイト受信が完了すると、RXI 割り込みが発生し、受信データレジスタ（RDR）の値を指定されたバッファに読み出します。

③ 1 バイト受信完了ごとに RXI 割り込みが発生し、RDR レジスタから受信データを読み出します。

④ 最終データ読み出し時の RXI 割り込みで、rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注 受信エラーが発生した場合、ERI 割り込みが発生します。割り込み処理にて発生したエラーを判別し、受信エラーがパリティエラーの場合は、エラーシグナルを出力します。受信エラーがオーバランエラーの場合は、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。
また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

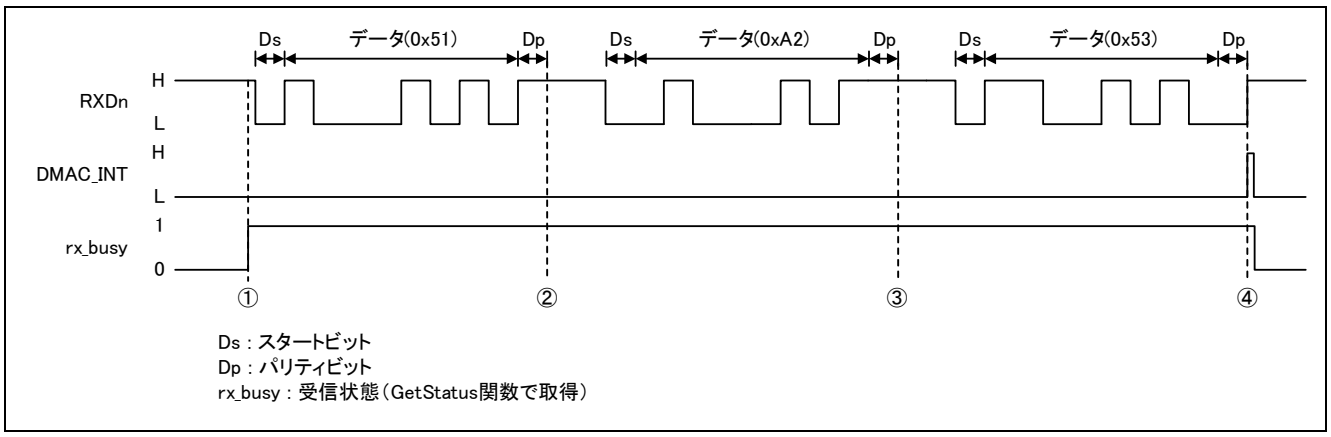


図 3-44 DMAC 制御による受信動作（3 バイト受信、偶数パリティ）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)になります。

② 1 バイト受信が完了すると、DMA 転送にて、受信データレジスタ（RDR）の値を指定されたバッファに転送します。

③ 1 バイト受信完了ごとに DMA 転送が発生し、RDR レジスタから受信データを読み出します。

④ 最終データ転送後の DMAC 転送完了割り込みで、rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注 受信エラーが発生した場合、ERI 割り込みが発生します。割り込み処理にて発生したエラーを判別し、受信エラーがパリティエラーの場合は、エラーシグナルを出力します。受信エラーがオーバランエラーの場合は、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。
また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

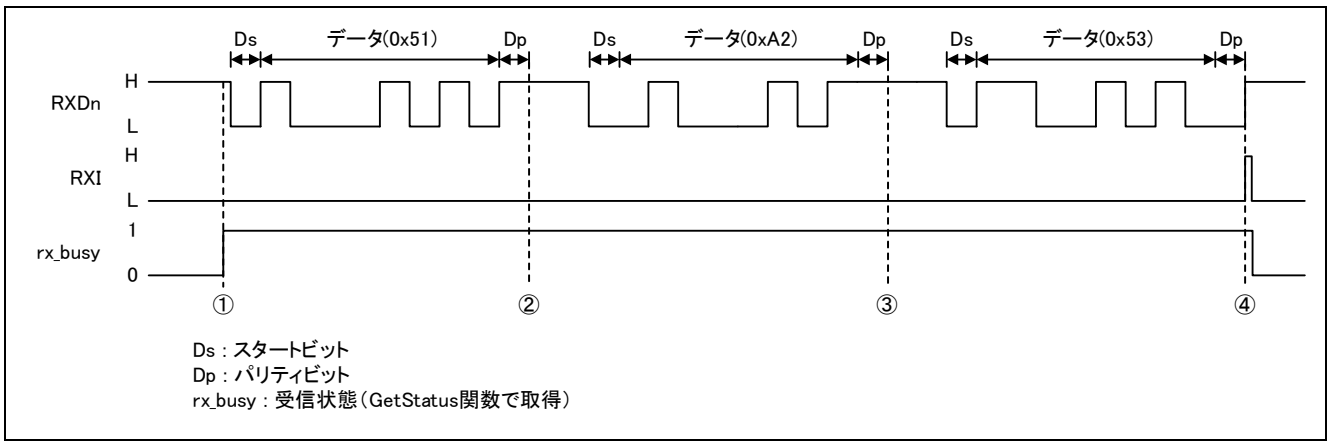


図 3-45 DTC 制御による受信動作（3 バイト受信、偶数パリティ）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)になります。

② 1 バイト受信が完了すると、DMA 転送にて、受信データレジスタ（RDR）の値を指定されたバッファに転送します。

③ 1 バイト受信完了ごとに DMA 転送が発生し、RDR レジスタから受信データを読み出します。

④ 最終データ転送後の RXI 割り込みで、rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注 受信エラーが発生した場合、ERI 割り込みが発生します。割り込み処理にて発生したエラーを判別し、受信エラーがパリティエラーの場合は、エラーシグナルを出力します。受信エラーがオーバランエラーの場合は、rx_busy フラグを”0”(受信待ち状態)にし、エラー状態をクリアします。
また、コールバック関数が登録されている場合、エラーイベント情報を引数にコールバック関数を実行します。

3.4.5 スマートカードモード受信時のエラーシグナル送信

スマートカードモードで受信中にパリティエラーを検出すると、H/W にて自動的にエラーシグナルを送信します。また、ERI 割り込みが発生し、コールバック関数が登録されている場合は、ARM_USART_EVENT_RX_PARITY_ERROR を引数にコールバック関数が呼び出されます。

送信中にパリティエラーを検出した場合の動作を図 3-46 に示します。

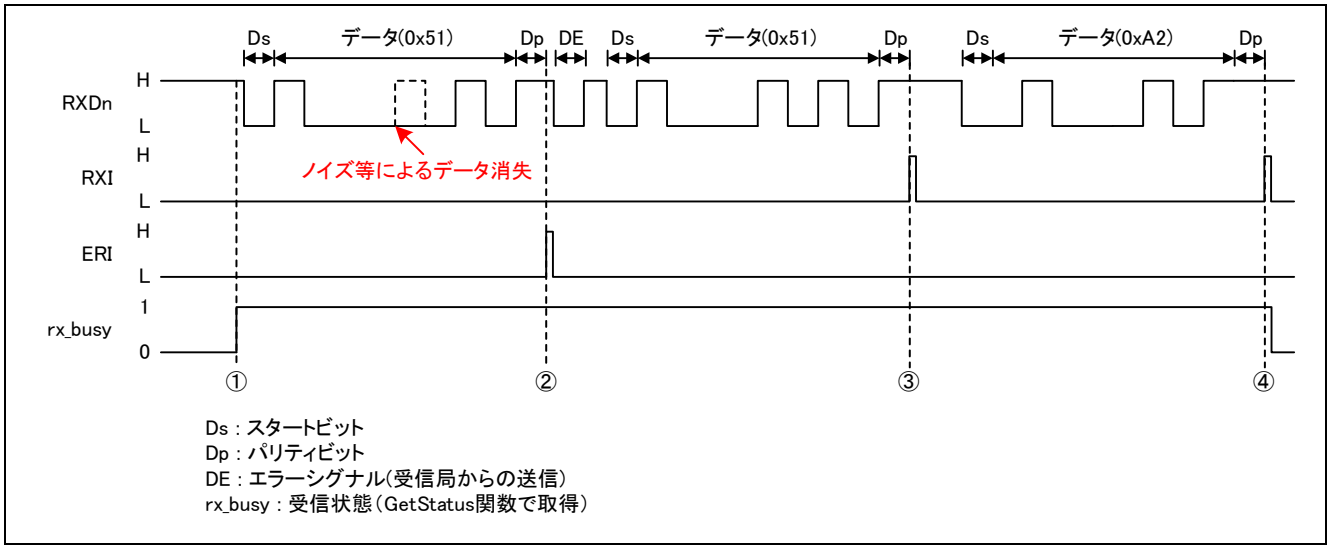


図 3-46 パリティエラー検出時の動作（2 バイト受信、偶数パリティ）

① Receive 関数を実行すると、rx_busy フラグが”1”(受信ビジー)になります。

② データ受信時にパリティエラーを検出すると ERI 割り込みが発生し、H/W にて自動的にエラーシグナルが出力されます。（受信したデータは破棄され、RXI 割り込み要求は発生しません）
また、コールバック関数が登録されている場合、ARM_USART_EVENT_RX_PARITY_ERROR を引数にコールバック関数を実行します。

③ 再送されたデータを受信しパリティエラーを検出しなかった場合、RXI 割り込みが発生します。割り込み処理にて、受信データレジスタ（RDR）の値を指定されたバッファに読み出します。

④ 最終データ読み出し時の RXI 割り込みで、rx_busy フラグを”0”(受信待ち状態)にします。また、コールバック関数が登録されている場合、ARM_USART_EVENT_RECEIVE_COMPLETE を引数にコールバック関数を実行します。(注)

注 最終データ呼び出し時にエラーシグナルを受信した場合も②と同様、ERI 割り込みの発生およびエラーシグナルの出力が行われます。

3.5 コンフィグレーション

USART ドライバは、ユーザが設定可能なコンフィグレーションを r_usart_cfg.h ファイルに用意します。

3.5.1 送信制御設定

送信制御方法を設定します。

名称：SCIn_TRANSMIT_CONTROL (n = 0～5、9)

表 3-7 SCIn_TRANSMIT_CONTROL の設定

設定値	内容
SCI_USED_INTERRUPT（初期値）	送信制御に割り込みを使用
SCI_USED_DMAC0	送信制御に DMAC0 を使用
SCI_USED_DMAC1	送信制御に DMAC1 を使用
SCI_USED_DMAC2	送信制御に DMAC2 を使用
SCI_USED_DMAC3	送信制御に DMAC3 を使用
SCI_USED_DTC	送信制御に DTC を使用

3.5.2 受信制御設定

受信制御方法を設定します。

名称：SCIn_RECEIVE_CONTROL (n = 0～5、9)

表 3-8 SCIn_RECEIVE_CONTROL の設定

設定値	内容
SCI_USED_INTERRUPT（初期値）	受信制御に割り込みを使用
SCI_USED_DMAC0	受信制御に DMAC0 を使用
SCI_USED_DMAC1	受信制御に DMAC1 を使用
SCI_USED_DMAC2	受信制御に DMAC2 を使用
SCI_USED_DMAC3	受信制御に DMAC3 を使用
SCI_USED_DTC	受信制御に DTC を使用

3.5.3 TDRE チェックタイムアウト時間

DMAC/DTC での送信開始時に送信バッファ空待ちのタイムアウト時間 (0～65535)

名称：SCI_CHECK_TDRE_TIMEOUT

初期値：(10)

3.5.4 TXI 割り込み優先レベル

TXIn 割り込みの優先レベルを設定します。(n=0～5、9)

名称：SCIn_TXI_PRIORITY

表 3-9 SCIn_TXI_PRIORITY の設定

設定値	内容
0	割り込み優先レベルを 0（最高）に設定
1	割り込み優先レベルを 1 に設定
2	割り込み優先レベルを 2 に設定
3（初期値）	割り込み優先レベルを 3（最低）に設定

3.5.5 RXI 割り込み優先レベル

RXIn 割り込みの優先レベルを設定します。(n=0～5、9)

名称：SCIn_RXI_PRIORITY

表 3-10 SCIn_RXI_PRIORITY の設定

設定値	内容
0	割り込み優先レベルを 0（最高）に設定
1	割り込み優先レベルを 1 に設定
2	割り込み優先レベルを 2 に設定
3（初期値）	割り込み優先レベルを 3（最低）に設定

3.5.6 ERI 割り込み優先レベル

ERIn 割り込みの優先レベルを設定します。(n=0～5、9)

名称：SCIn_ERI_PRIORITY

表 3-11 SCIn_ERI_PRIORITY の設定

設定値	内容
0	割り込み優先レベルを 0（最高）に設定
1	割り込み優先レベルを 1 に設定
2	割り込み優先レベルを 2 に設定
3（初期値）	割り込み優先レベルを 3（最低）に設定

3.5.7 ソフトウェア制御による CTS 端子定義

ソフトウェア制御にて使用する CTS 端子を定義します。

名称：USARTn_CTS_PORT(注)、USARTn_CTS_PIN(n=0～5、9)

表 3-12 USARTn_CTS_PORT、USARTn_CTS_PIN の設定

名称	初期値	内容
USARTn_CTS_PORT(注 1)	(PORT0->PIDR)	CTS 端子に PORT0 を選択
USARTn_CTS_PIN	0	CTS 端子に PORTi00 を選択 (I は USARTn_CTS_PORT で指定したポート)

注　デフォルトでは本定義はコメントアウトしています。
ソフトウェア制御による CTS 端子を使用する場合は、コメントアウトを解除してください。

3.5.8 ソフトウェア制御による RTS 端子定義

ソフトウェア制御にて使用する RTS 端子を定義します。

名称：USARTn_RTS_PORT(注 2)、USARTn_RTS_PIN(n=0～5、9)

表 3-13 USARTn_RTS_PORT、USARTn_RTS_PIN の設定

名称	初期値	内容
USARTn_RTS_PORT(注)	(PORT0->PODR)	RTS 端子に PORT0 を選択
USARTn_RTS_PIN	0	RTS 端子に PORTi00 を選択 (I は USARTn_RTS_PORT で指定したポート)

注　デフォルトでは本定義はコメントアウトしています。
ソフトウェア制御による RTS 端子を使用する場合は、コメントアウトを解除してください。

3.5.9 関数の RAM 配置

USART ドライバの特定関数を RAM で実行するための設定を行います。

関数の RAM 配置を設定するコンフィグレーションは、関数ごとに定義を持ちます。

名称：USART_CFG_SECTION_xxx

xxx には関数名をすべて大文字で記載

例） ARM_USART_INITIALIZE 関数 → USART_CFG_SECTION_ARM_USART_INITIALIZE

表 3-14 USART_CFG_SECTION_xxx の設定

設定値	内容
SYSTEM_SECTION_CODE	関数を RAM に配置しません
SYSTEM_SECTION_RAM_FUNC	関数を RAM に配置します

表 3-15 各関数の RAM 配置初期状態

番号	関数名	RAM 配置
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=0～5、9）（TXI 割り込み処理）	✔
16	scin_rxi_interrupt（n=0～5、9）（RXI 割り込み処理）	✔
17	scin_eri_interrupt（n=0～5、9）（ERI 割り込み処理）	✔

4. ドライバ詳細情報

本章では、本ドライバ機能を構成する詳細仕様について説明します。

4.1 関数仕様

USART ドライバの各関数の仕様と処理フローを示します。

処理フロー内では条件分岐などの判定方法の一部を省略して記述しているため、実際の処理と異なる場合があります。

4.1.1 ARM_USART_GetVersion 関数

表 4-1 ARM_USART_GetVersion 関数仕様

書式	ARM_DRIVER_VERSION ARM_USART_GetVersion(void)
仕様説明	USART ドライバのバージョンを取得します
引数	なし
戻り値	USART ドライバのバージョン

備考	[インスタンスからの関数呼び出し例]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	ARM_DRIVER_VERSION version;
	version = sci0Drv->GetVersion();
	
	}

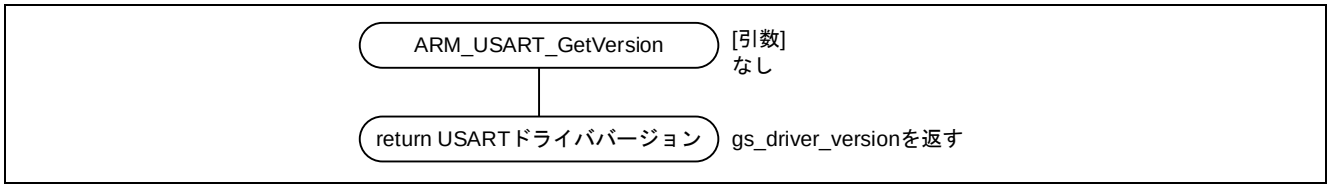


図 4-1 ARM_USART_GetVersion 関数処理フロー

4.1.2 ARM_USART_GetCapabilities 関数

表 4-2 ARM_USART_GetCapabilities 関数仕様

書式	ARM_USART_CAPABILITIES ARM_USART_GetCapabilities(void)
仕様説明	USART ドライバの機能を取得します
引数	なし
戻り値	ドライバ機能

備考	[インスタンスからの関数呼び出し例]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	ARM_USART_CAPABILITIES cap;
	cap = sci0Drv->GetCapabilities();
	
	}

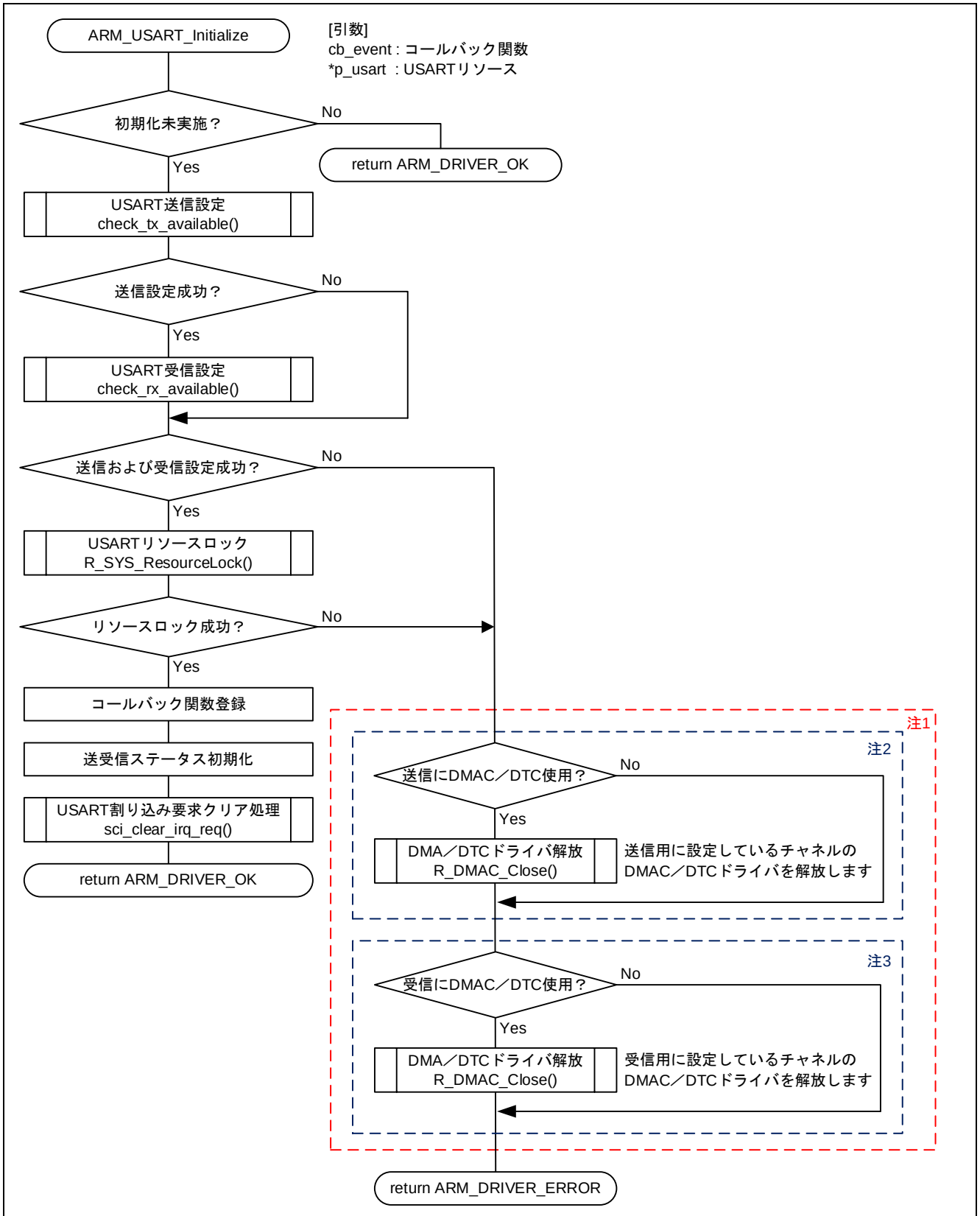
図 4-2 ARM_USART_GetCapabilities 関数処理フロー

4.1.3 ARM_USART_Initialize 関数

表 4-3 ARM_USART_Initialize 関数仕様

書式	int32_t ARM_USART_Initialize(ARM_USART_SignalEvent_t cb_event, st_usart_resources_t * const p_usart)
仕様説明	USART ドライバの初期化（RAM の初期化、レジスタ設定、NVIC への登録）を行います
引数	ARM_USART_SignalEvent_t cb_event: コールバック関数 イベント発生時のコールバック関数を指定します。NULL を設定した場合、コールバック関数が実行されません。
	st_usart_resources_t * const p_usart : USART のリソース 初期化する USART のリソースを指定します。
戻り値	ARM_DRIVER_OK　　USART の初期化成功
	ARM_DRIVER_ERROR　　USART の初期化失敗 以下のいずれかの状態を検出すると初期化失敗となります ・送信、受信ともに使用不可の場合（通信制御設定、NVIC 登録設定などに不備がある場合） ・使用する SCI チャネルのリソースがロックされている場合 （すでに R_SYS_ResourceLock 関数にて SCIn がロックされている場合）

備考	インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	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);
	}



注1. r_usart_cfg.hにて、いずれかのチャネルでDMAC／DTC機能を使用する設定にした場合、該当の処理が有効になります
注2. r_usart_cfg.hにて、いずれかのチャネルの送信でDMAC／DTC機能を使用する設定にした場合、該当の処理が有効になります
注3. r_usart_cfg.hにて、いずれかのチャネルの受信でDMAC／DTC機能を使用する設定にした場合、該当の処理が有効になります

図 4-3 ARM_USART_Initialize 関数処理フロー

4.1.4 ARM_USART_Uninitialize 関数

表 4-4 ARM_USART_Uninitialize 関数仕様

書式	int32_t ARM_USART_Uninitialize(st_usart_resources_t * const p_usart)
仕様説明	USART ドライバを解放します
引数	st_usart_resources_t * const p_usart: USART のリソース 解放する USART のリソースを指定します。
戻り値	ARM_DRIVER_OK　　USART の解放成功

備考	インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	sci0Drv->Uninitialize();
	}

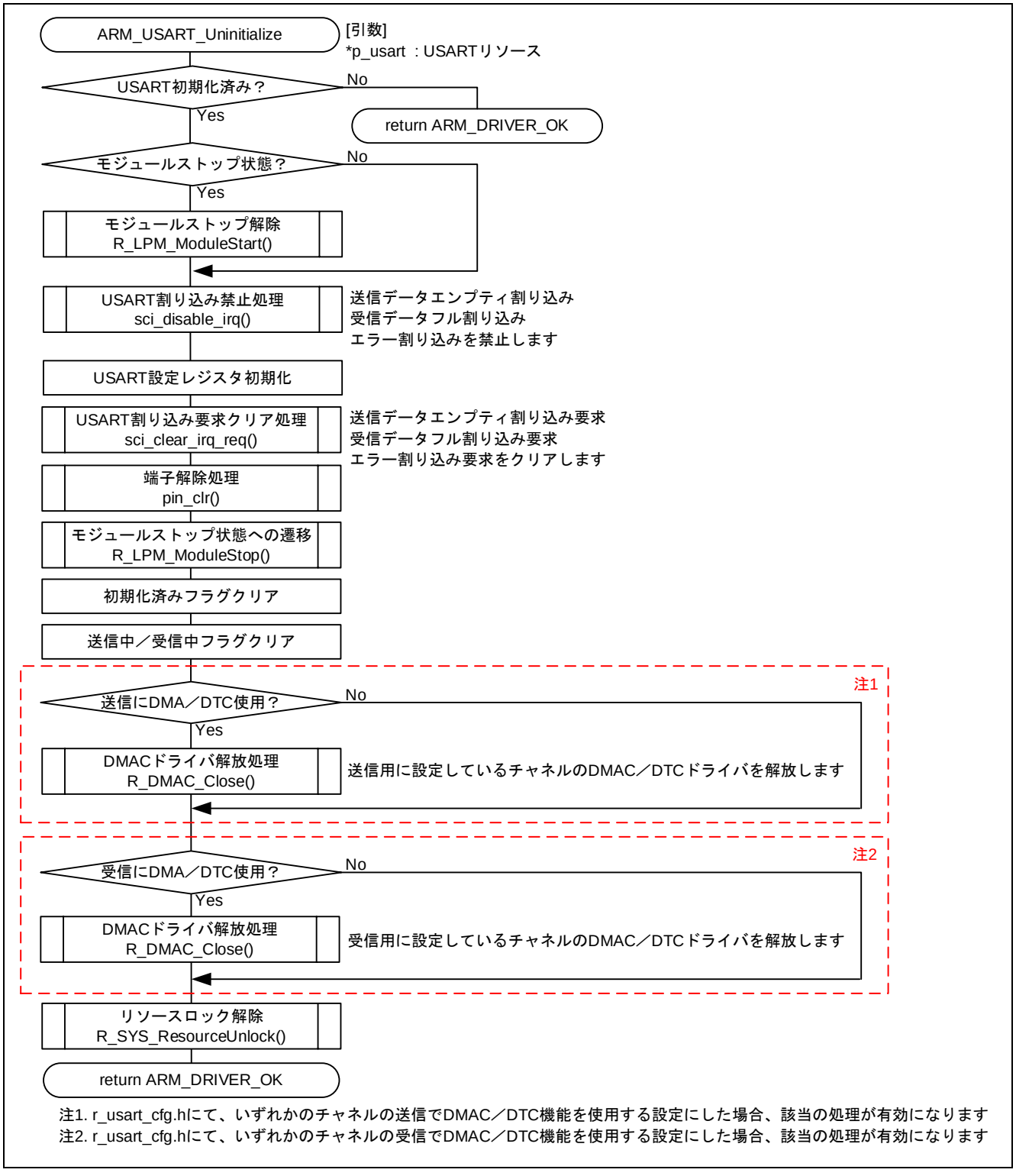


図 4-4 ARM_USART_Uninitailze 関数処理フロー

4.1.5 ARM_USART_PowerControl 関数

表 4-5 ARM_USART_PowerControl 関数仕様

書式	int32_t ARM_USART_PowerControl(ARM_POWER_STATE state, st_usart_resources_t * const p_usart)
仕様説明	USART のモジュールストップ状態の解除または遷移を行います
引数	ARM_POWER_STATE state : 電力設定 以下のいずれかを設定します ARM_POWER_OFF : モジュールストップ状態に遷移します ARM_POWER_FULL : モジュールストップ状態を解除します ARM_POWER_LOW : 本設定はサポートしておりません
	st_usart_resources_t * const p_usart: USART のリソース 電源供給する USART のリソースを指定します。
戻り値	ARM_DRIVER_OK 電力設定変更成功
	ARM_DRIVER_ERROR 電力設定変更失敗 以下のいずれかの条件を検出すると電力設定変更失敗となります ・USART の未初期化状態で実行した場合 ・モジュールストップの遷移に失敗した場合（R_LPM_ModuleStart にてエラーが発生した場合）
	ARM_DRIVER_ERROR_UNSUPPORTED サポート外の電力設定を指定

備考	インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	sci0Drv->PowerControl(ARM_POWER_FULL);
	}

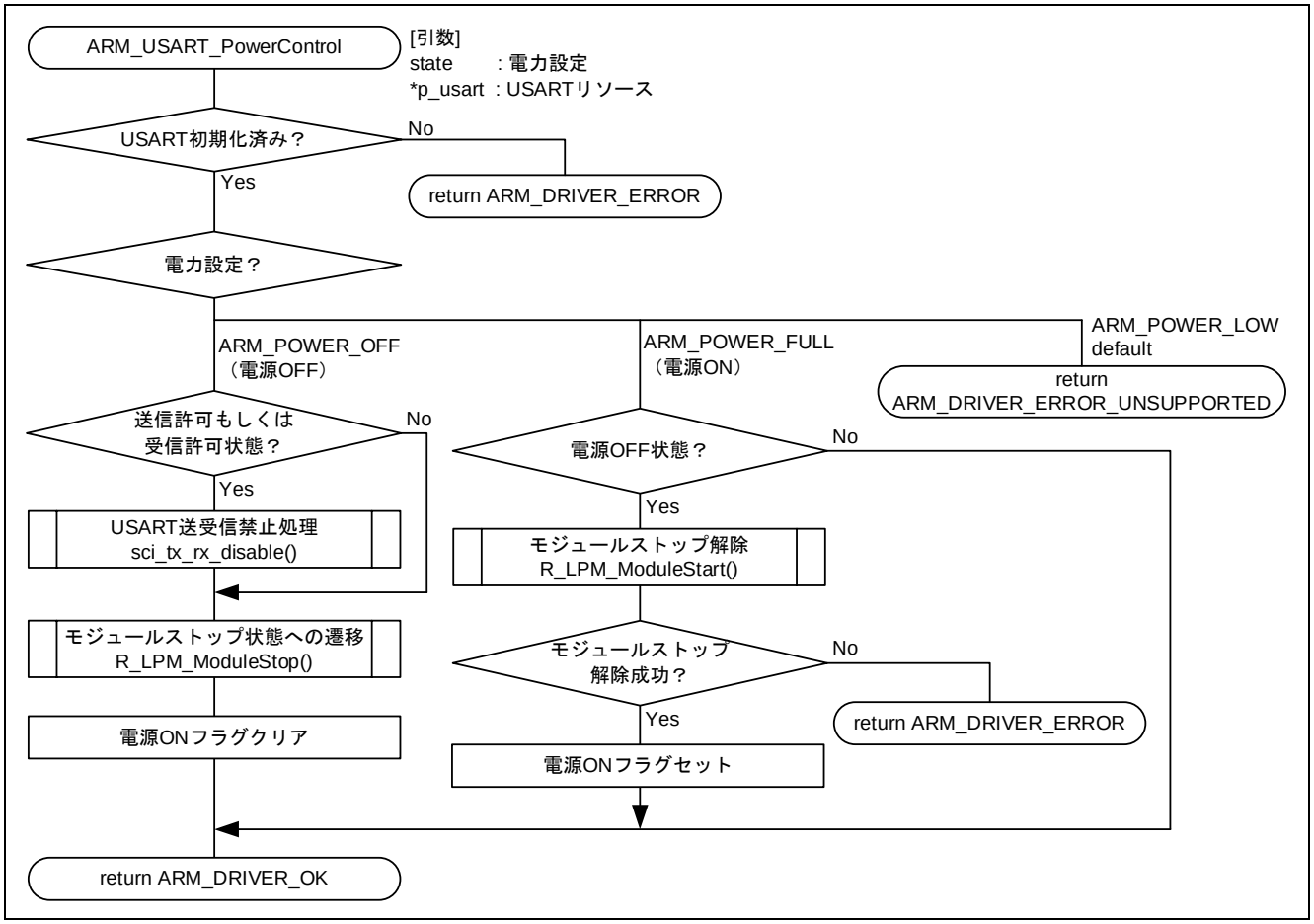


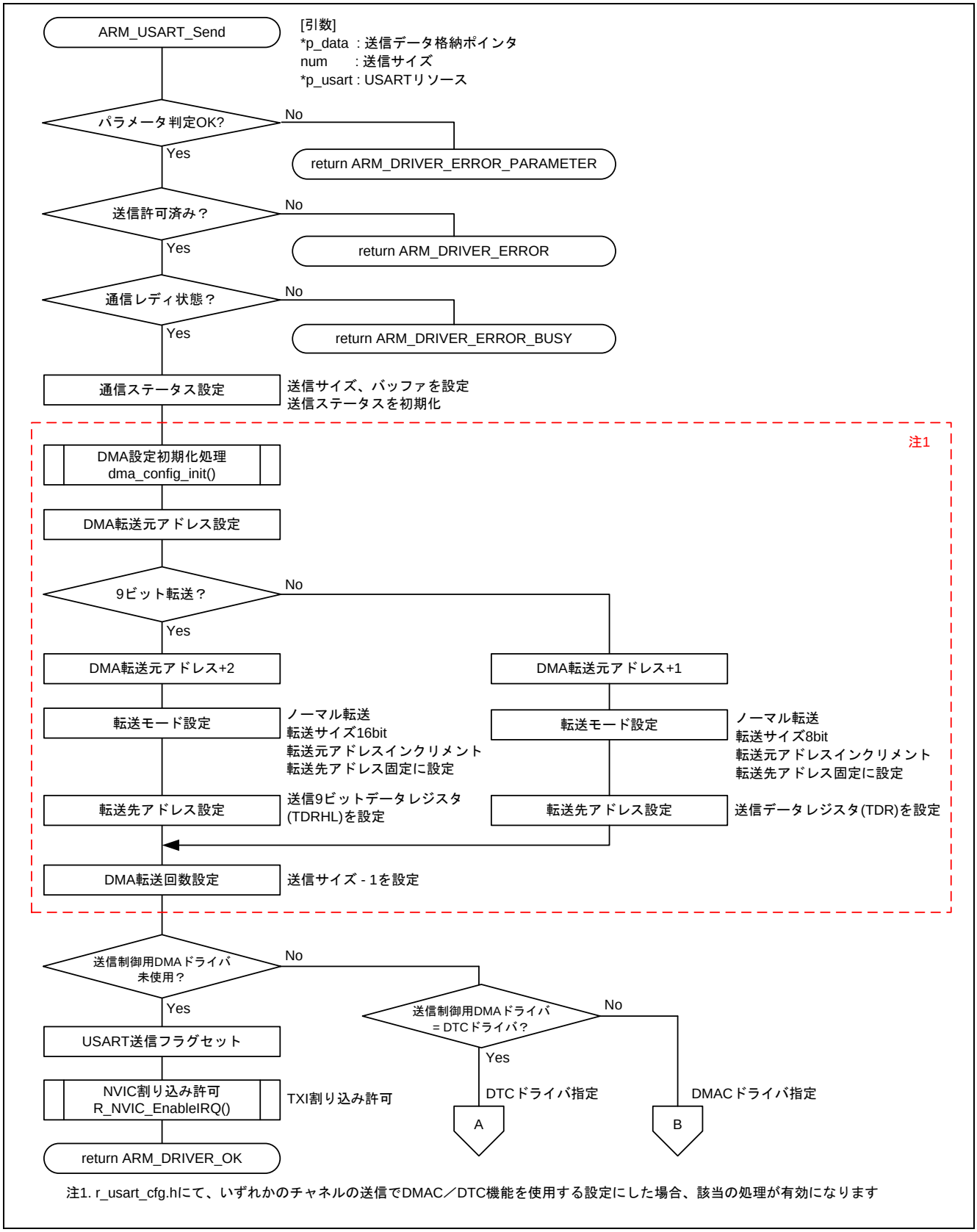
図 4-5 ARM_USART_PowerControl 関数処理フロー

4.1.6 ARM_USART_Send 関数

表 4-6 ARM_USART_Send 関数仕様

書式	int32_t ARM_USART_Send(void const * const p_data, uint32_t num, st_usart_resources_t * const p_usart)
仕様説明	送信を開始します
引数	void const * const *p_data : 送信データ格納ポインタ 送信するデータを格納したバッファの先頭アドレスを指定します
	uint32_t num : 送信サイズ 送信するデータサイズを指定します
	st_usart_resources_t * const p_usart : USART のリソース 送信する USART のリソースを指定します。
戻り値	ARM_DRIVER_OK 送信開始成功
	ARM_DRIVER_ERROR 送信開始失敗 以下のいずれかの状態を検出すると送信開始失敗となります ・送信禁止状態で実行した場合 ・送信処理に DMAC/DTC を指定した状態で、DMA ドライバの設定に失敗した場合
	ARM_DRIVER_ERROR_BUSY ビジー状態による送信失敗 以下のいずれかの状態を検出するとビジー状態による送信失敗となります ・送信中判定 ・クロック同期モードで受信中判定 ・送信処理に DMAC/DTC を使用時、送信バッファ空待ちのタイムアウトが発生した場合
	ARM_DRIVER_ERROR_PARAMETER パラメータエラー 以下のいずれかの設定を行った場合、パラメータエラーとなります ・送信データ格納ポインタに NULL を設定した場合 ・送信データサイズに 0 を設定した場合

備考	インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	// 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);
	}



注1. r_usart_cfg.hにて、いずれかのチャネルの送信でDMAC／DTC機能を使用する設定にした場合、該当の処理が有効になります

図 4-6 ARM_USART_Send 関数処理フロー(1/3)

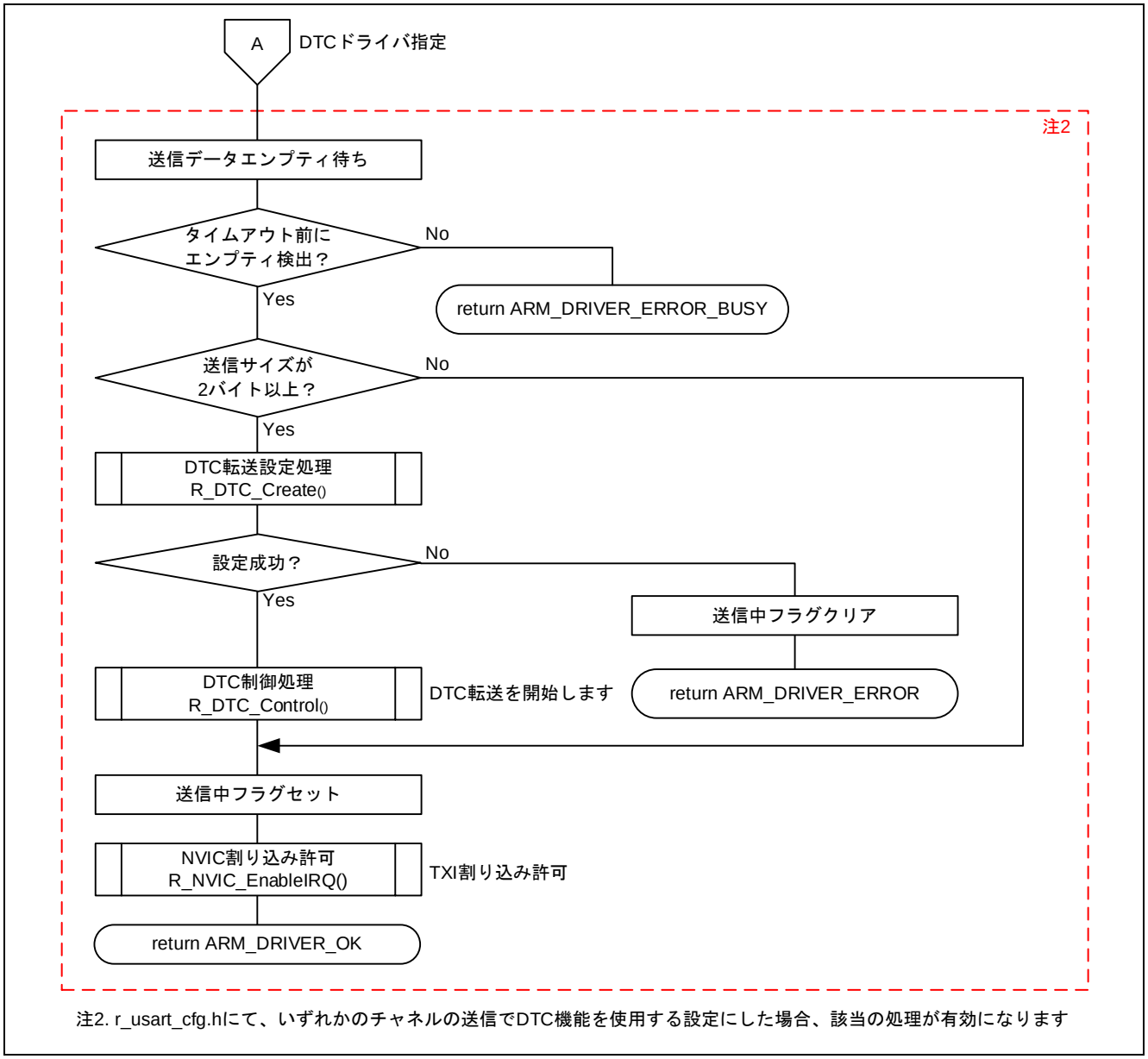
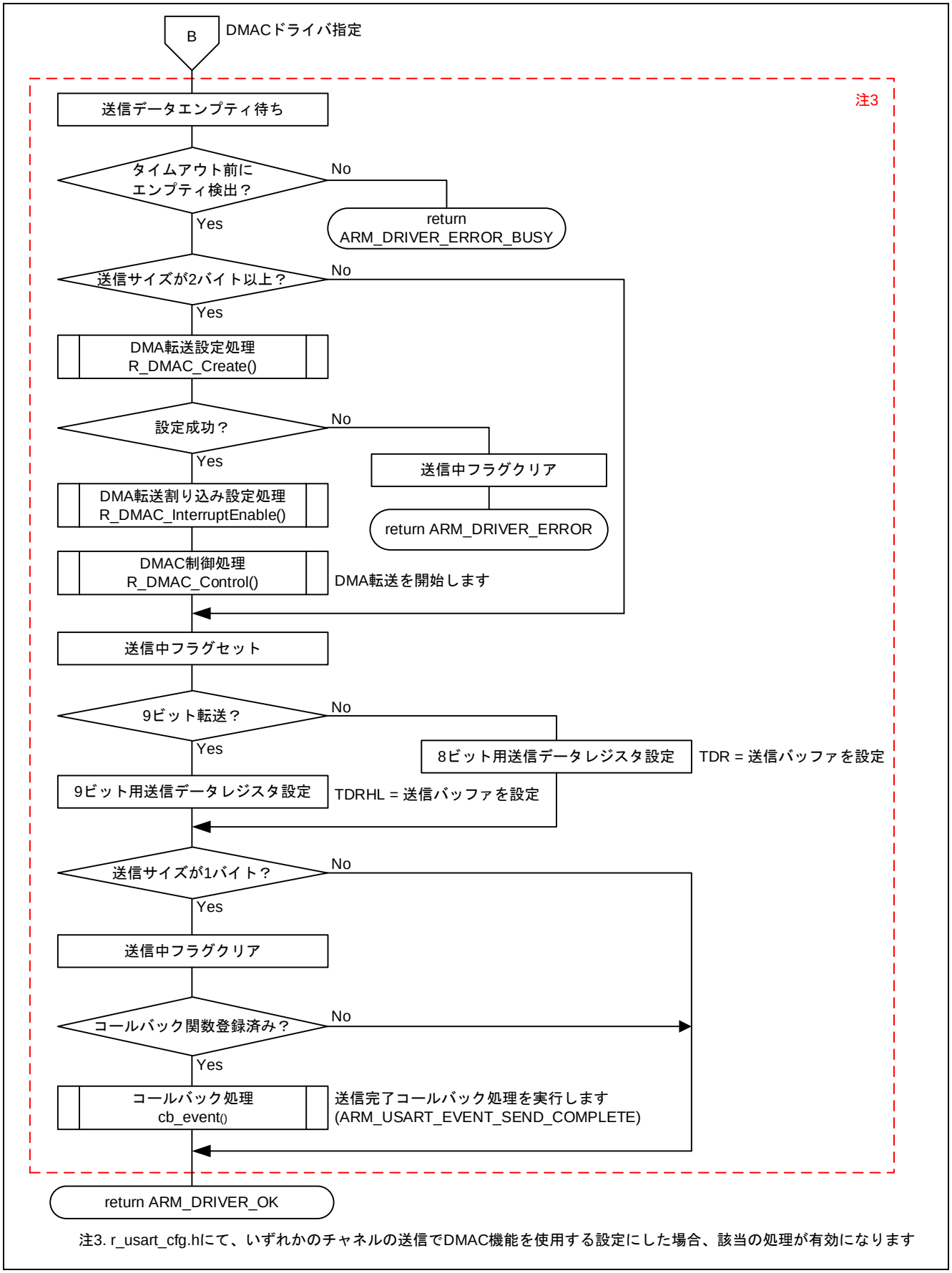


図 4-7 ARM_USART_Send 関数処理フロー(2/3)



注3. r_usart_cfg.hにて、いずれかのチャネルの送信でDMAC機能を使用する設定にした場合、該当の処理が有効になります

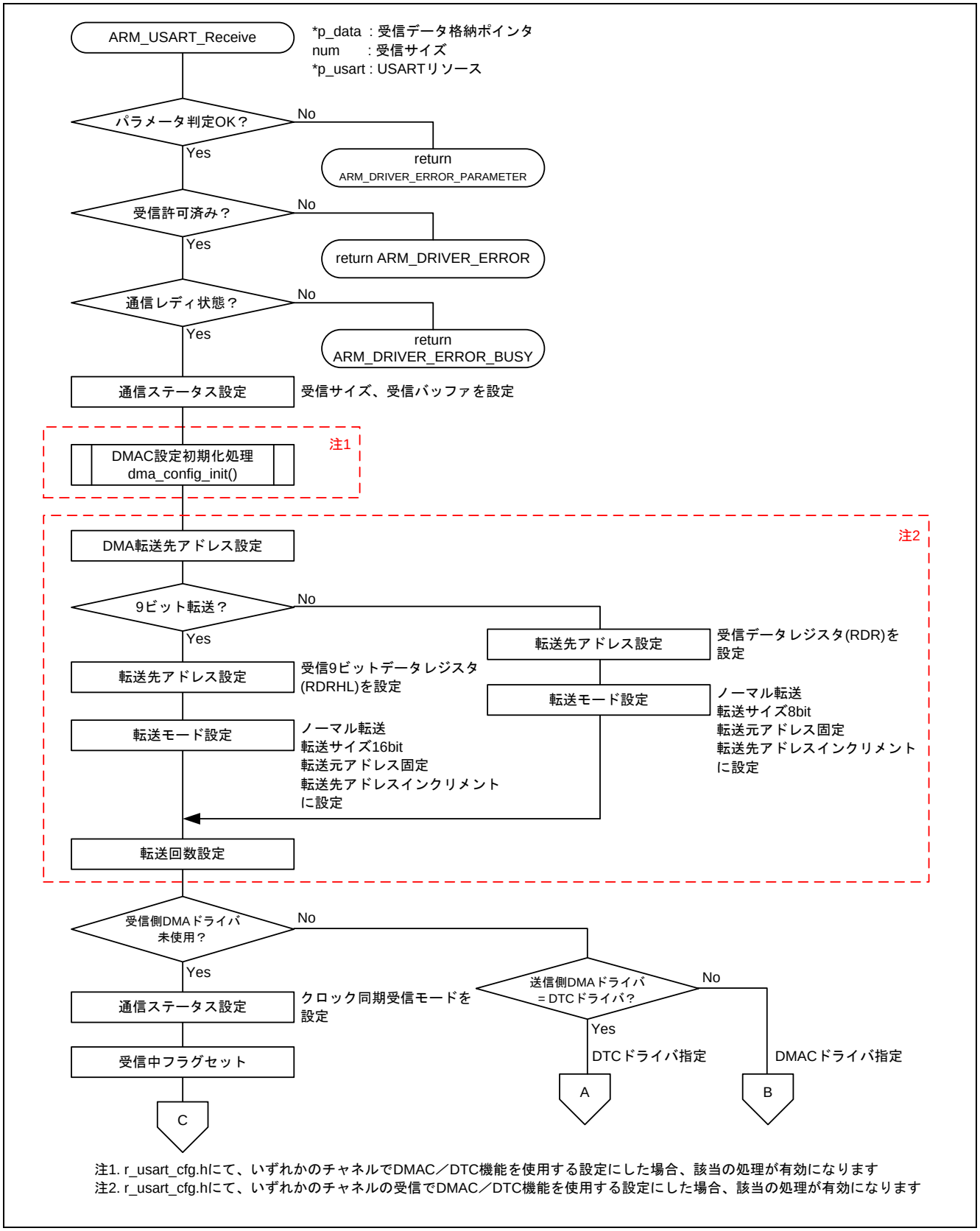
図 4-8 ARM_USART_Send 関数処理フロー(3/3)

4.1.7 ARM_USART_Receive 関数

表 4-7 ARM_USART_Receive 関数仕様

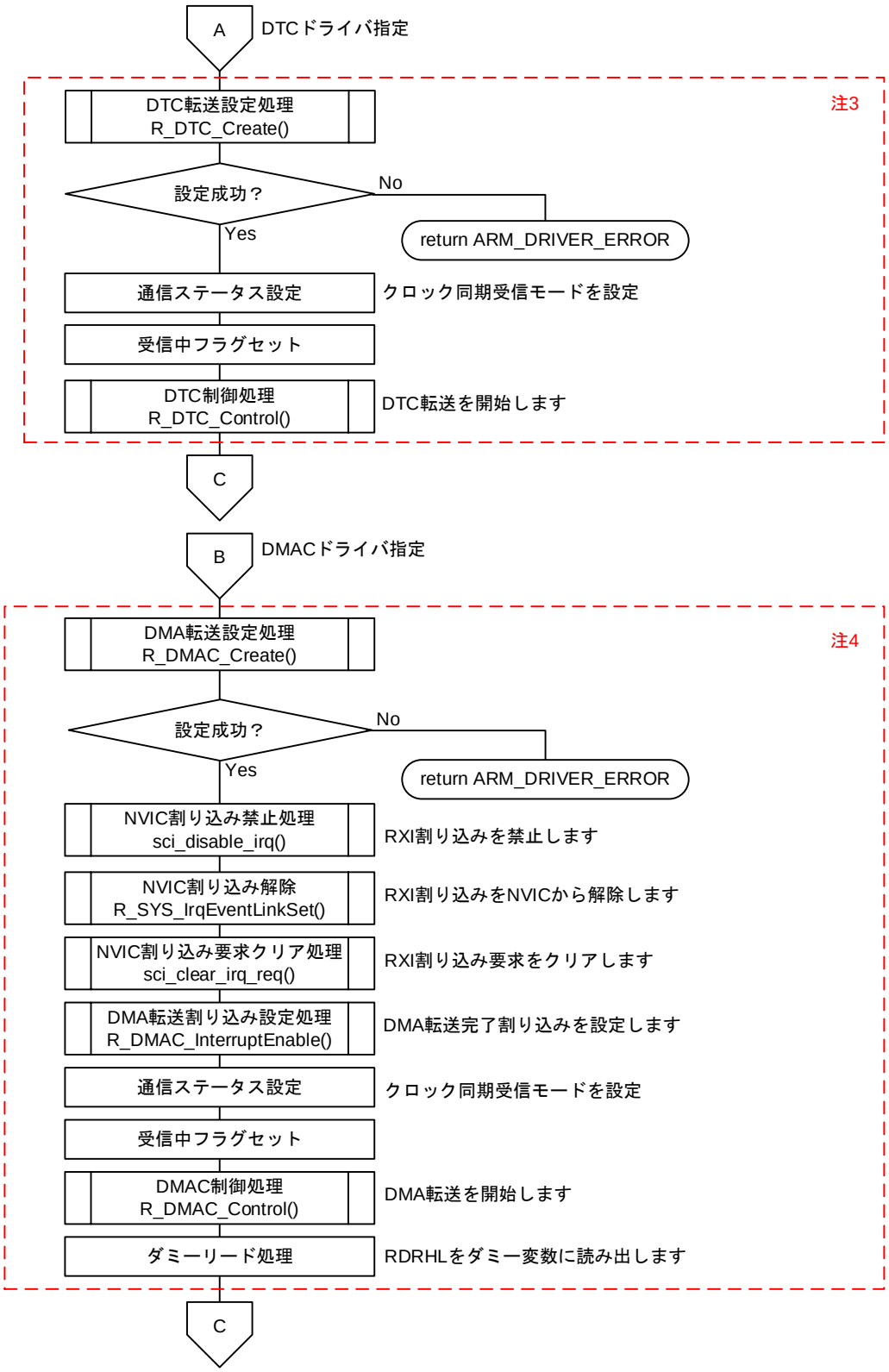
書式	int32_t ARM_USART_Receive(void * const p_data, uint32_t num, st_usart_resources_t * const p_usart)
仕様説明	受信を開始します
引数	void * const p_data: 受信データ格納ポインタ 受信したデータを格納するバッファの先頭アドレスを指定します
	uint32_t num : 受信サイズ 受信するデータサイズを指定します
	st_usart_resources_t * const p_usart : USART のリソース 受信する USART のリソースを指定します。
戻り値	ARM_DRIVER_OK 受信開始成功
	ARM_DRIVER_ERROR 受信開始失敗 以下のいずれかの状態を検出すると受信開始失敗となります ・受信禁止状態で実行した場合 ・受信処理に DMAC/DTC を指定した状態で、DMA ドライバの設定に失敗した場合
	ARM_DRIVER_ERROR_BUSY ビジー状態による受信失敗 以下のいずれかの状態を検出するとビジー状態による受信失敗となります ・受信中判定 ・クロック同期モードで送信中判定 ・送信処理に DMAC/DTC を使用、かつクロック同期モードにて、送信バッファ空待ちのタイムアウトが発生した場合
	ARM_DRIVER_ERROR_PARAMETER パラメータエラー 以下のいずれかの設定を行った場合、パラメータエラーとなります ・受信データ格納ポインタに NULL を設定した場合 ・受信データサイズに 0 を設定した場合

備考	インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	// 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);
	}



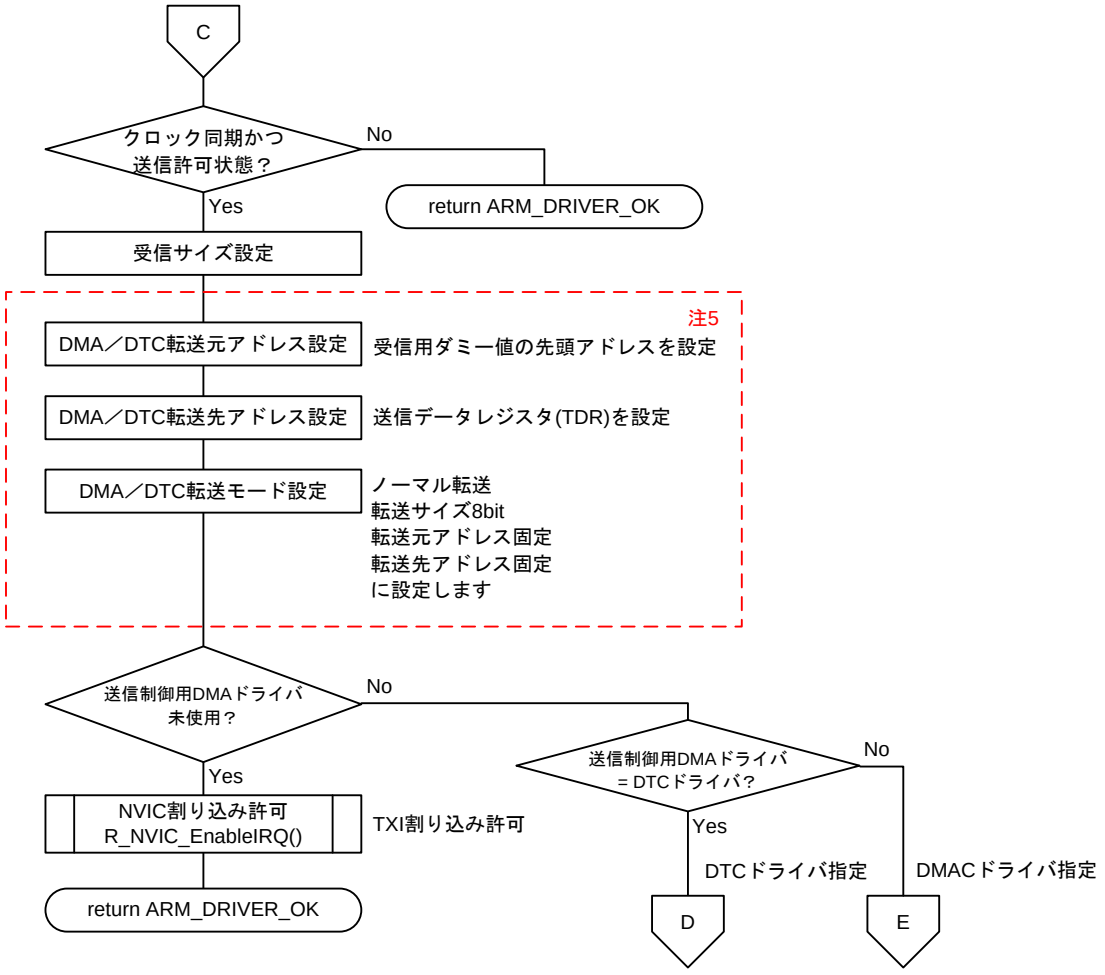
注1. r_usart_cfg.hにて、いずれかのチャネルでDMAC／DTC機能を使用する設定にした場合、該当の処理が有効になります
注2. r_usart_cfg.hにて、いずれかのチャネルの受信でDMAC／DTC機能を使用する設定にした場合、該当の処理が有効になります

図 4-9 ARM_USART_Receive 関数処理フロー(1/5)



注3. r_usart_cfg.hにて、いずれかのチャネルの受信でDTC機能を使用する設定にした場合、該当の処理が有効になります
注4. r_usart_cfg.hにて、いずれかのチャネルの受信でDMAC機能を使用する設定にした場合、該当の処理が有効になります

図 4-10 ARM_USART_Receive 関数処理フロー(2/5)



注5. r_usart_cfg.hにて、いずれかのチャネルの受信でDMAC／DTC機能を使用する設定にした場合、該当の処理が有効になります

図 4-11 ARM_USART_Receive 関数処理フロー(3/5)

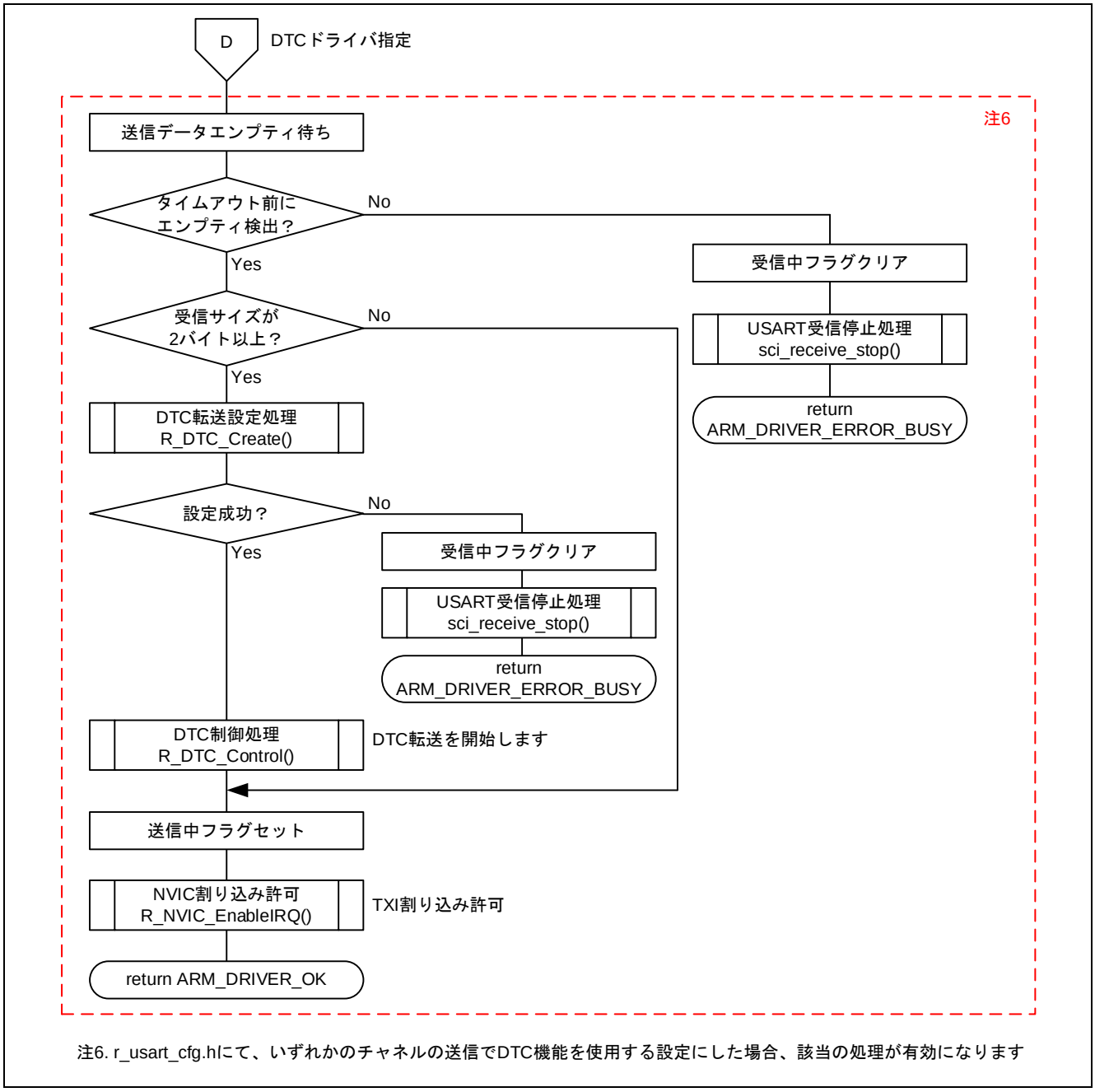


図 4-12 ARM_USART_Receive 関数処理フロー(4/5)

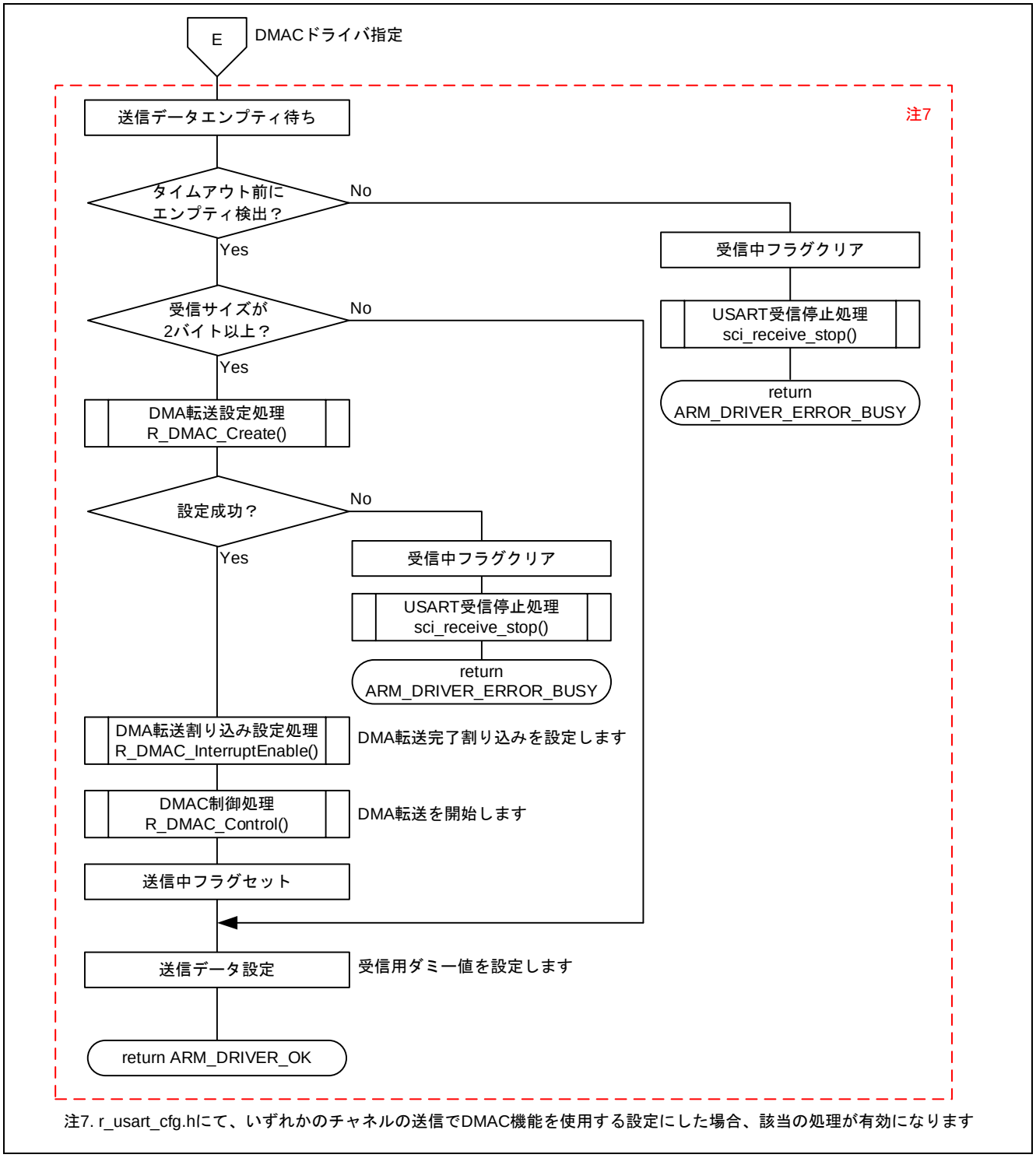


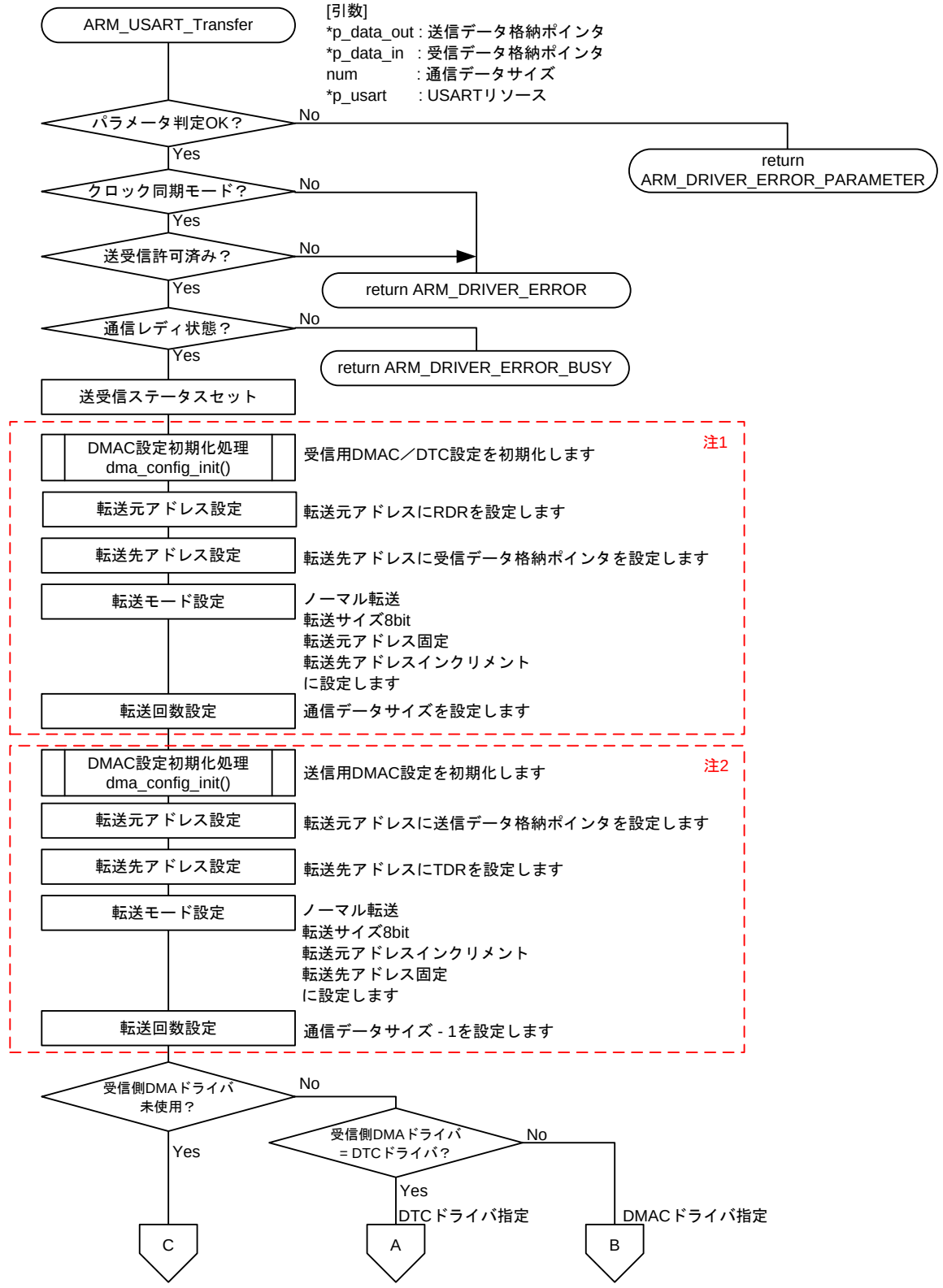
図 4-13 ARM_USART_Receive 関数処理フロー(5/5)

4.1.8 ARM_USART_Transfer 関数

表 4-8 ARM_USART_Transfer 関数仕様

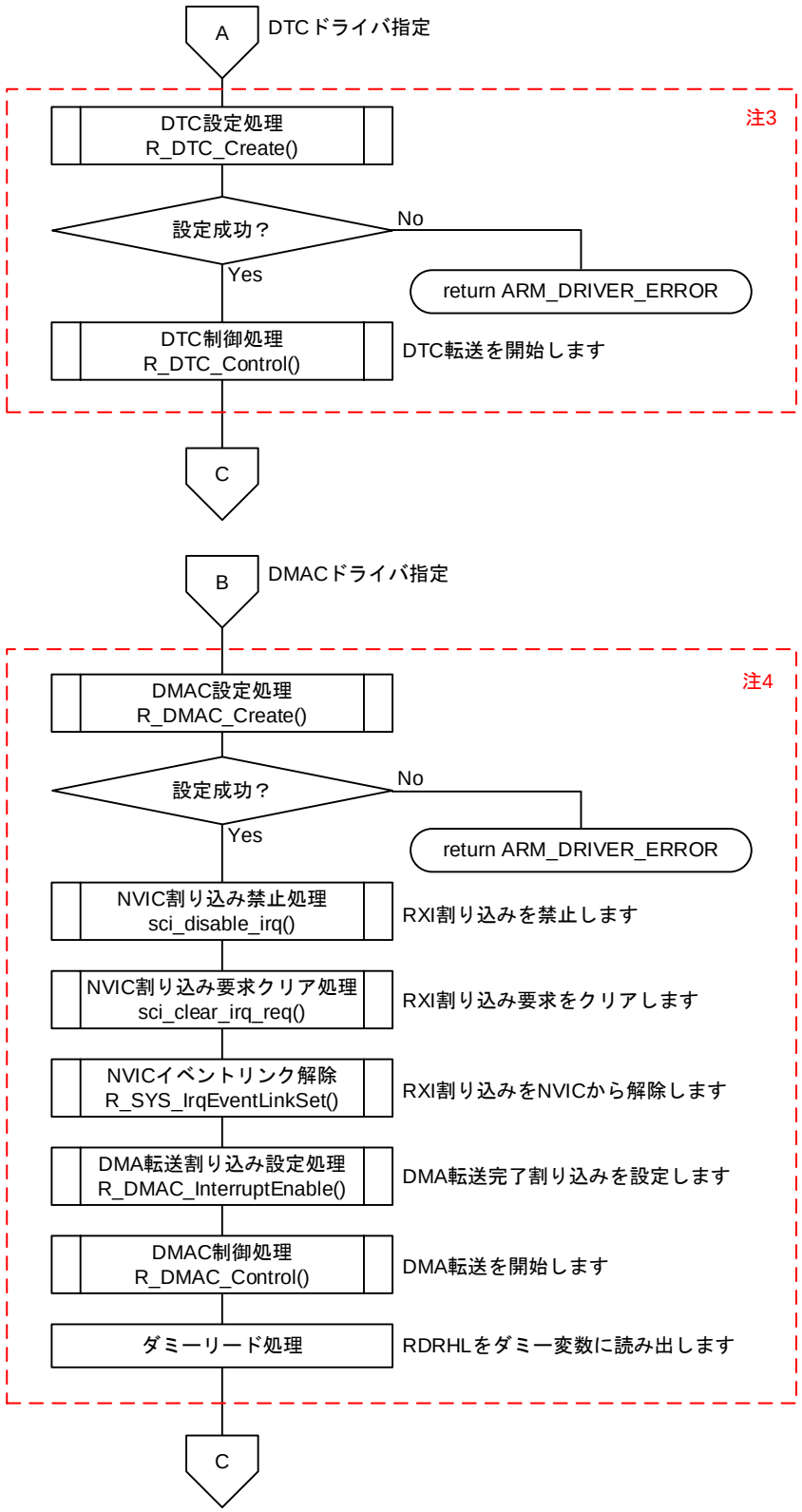
書式	int32_t ARM_USART_Transfer(void const * const p_data_out, void * const p_data_in, uint32_t num, st_usart_resources_t * const p_usart)
仕様説明	クロック同期モードの送受信を開始します
引数	void const * const p_data_out : 送信データ格納ポインタ 送信するデータを格納したバッファの先頭アドレスを指定します
	void * const p_data_in : 受信データ格納ポインタ 受信したデータを格納するバッファの先頭アドレスを指定します
	uint32_t num : 送受信サイズ 送受信するデータサイズを指定します
	st_usart_resources_t * const p_usart : USART のリソース 受信する USART のリソースを指定します。
戻り値	ARM_DRIVER_OK 送受信開始成功
	ARM_DRIVER_ERROR 送受信開始失敗 以下のいずれかの状態を検出すると送受信開始失敗となります ・クロック同期モードでない場合 ・送信禁止、または受信禁止状態で実行した場合 ・送信または受信処理に DMAC/DTC を指定した状態で、DMA ドライバの設定に失敗した場合
	ARM_DRIVER_ERROR_BUSY ビジー状態による送信失敗 以下のいずれかの状態を検出するとビジー状態による受信失敗となります ・受信中、または送信中判定 ・送信処理に DMAC/DTC を使用時、送信バッファ空待ちのタイムアウトが発生した場合
	ARM_DRIVER_ERROR_PARAMETER パラメータエラー 以下のいずれかの状態を検出するとパラメータエラーとなります ・送受信サイズが 0 の場合 ・送信データ格納ポインタが NULL の場合 ・受信データ格納ポインタが NULL の場合

備考	インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	// 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);
	}



注1. r_usart_cfg.hにて、いずれかのチャネルの受信でDMAC／DTC機能を使用する設定にした場合、該当の処理が有効になります
注2. r_usart_cfg.hにて、いずれかのチャネルの送信でDMAC／DTC機能を使用する設定にした場合、該当の処理が有効になります

図 4-14 ARM_USART_Transfer 関数処理フロー(1/4)



注3. r_usart_cfg.hにて、いずれかのチャネルの受信でDTC機能を使用する設定にした場合、該当の処理が有効になります
注4. r_usart_cfg.hにて、いずれかのチャネルの送信でDMAC機能を使用する設定にした場合、該当の処理が有効になります

図 4-15 ARM_USART_Transfer 関数処理フロー(2/4)

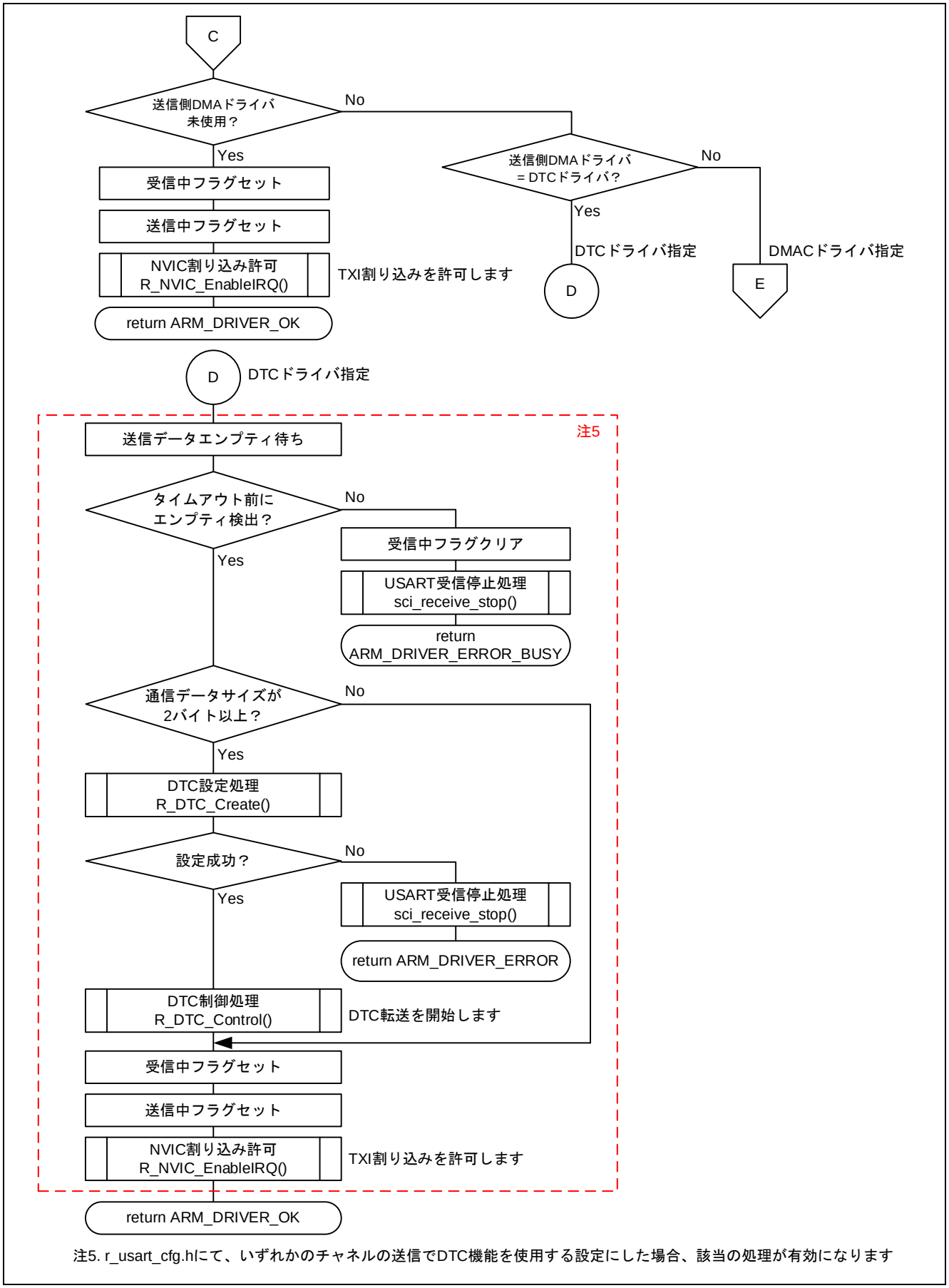


図 4-16 ARM_USART_Transfer 関数処理フロー(3/4)

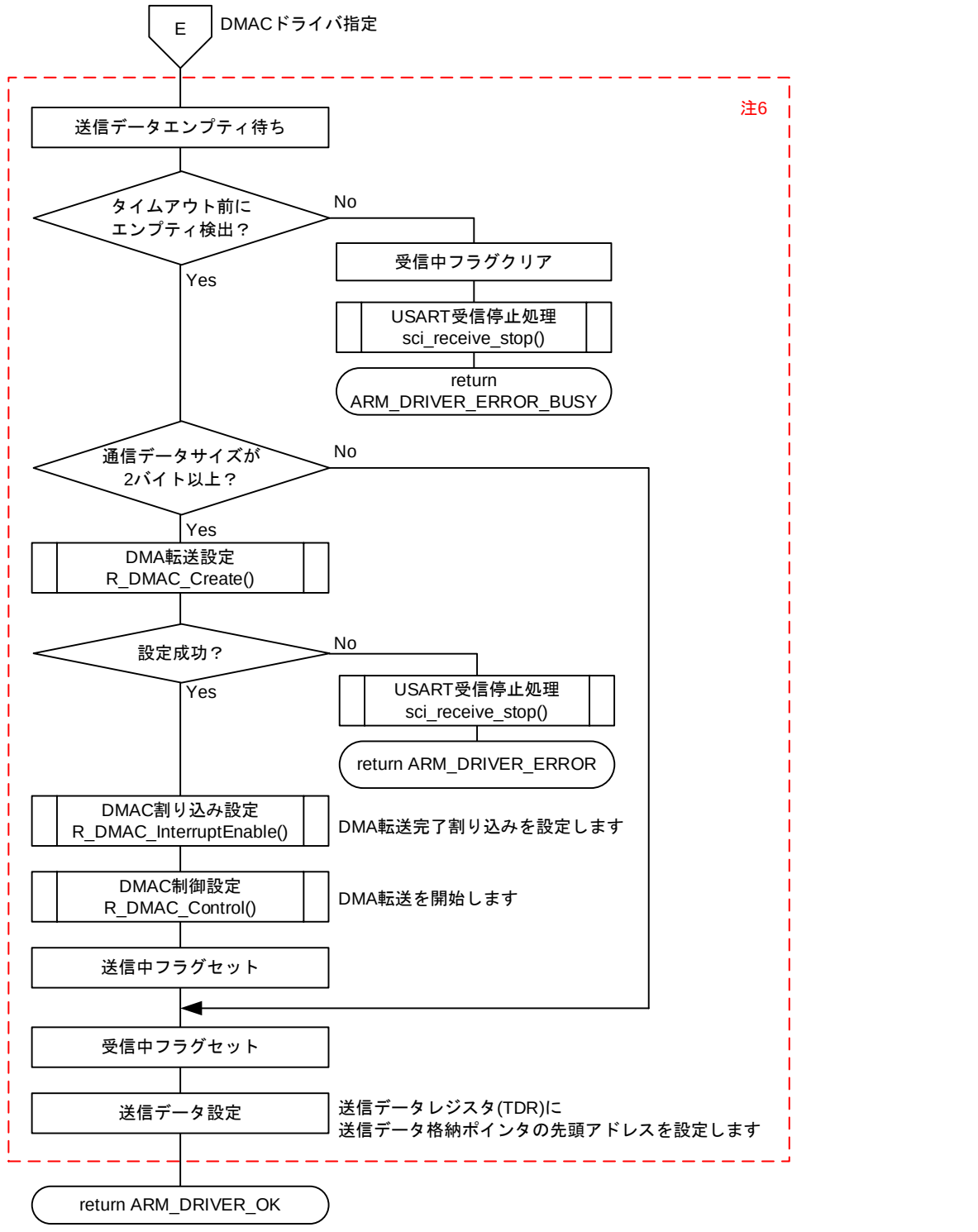


図 4-17 ARM_USART_Transfer 関数処理フロー(4/4)

4.1.9 ARM_USART_GetTxCount 関数

表 4-9 ARM_USART_GetTxCount 関数仕様

書式	uint32_t ARM_USART_GetTxCount(st_usart_resources_t const * const p_usart)
仕様説明	現時点の送信数を取得します
引数	st_usart_resources_t * const p_usart : USART のリソース 送信数を取得する USART のリソースを指定します。
戻り値	送信数

備考	インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	// 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();
	}

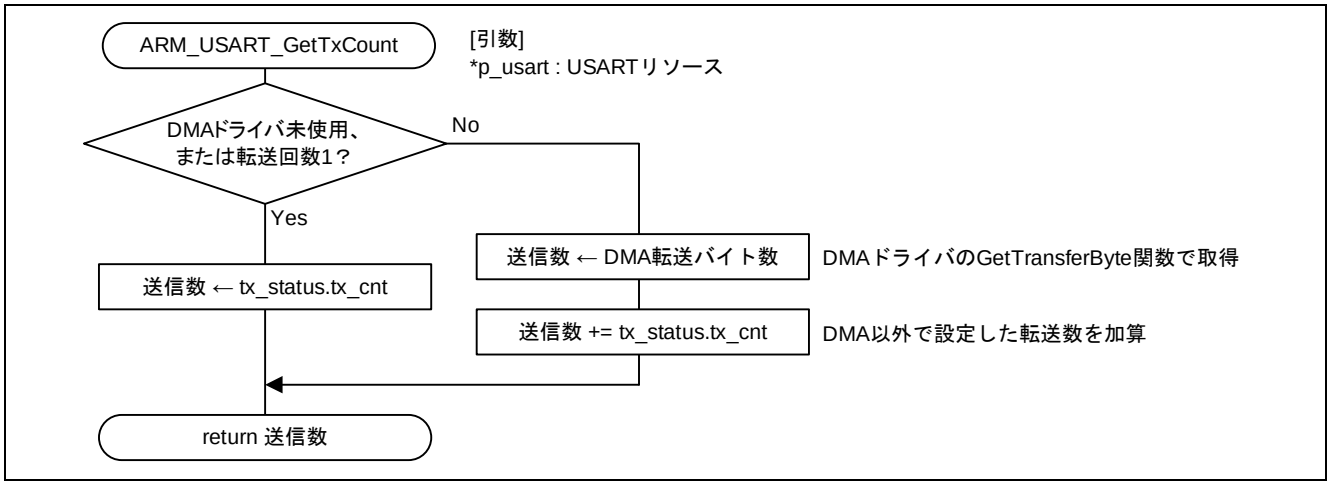


図 4-18 ARM_USART_GetTxCount 関数処理フロー

4.1.10 ARM_USART_GetRxCount 関数

表 4-10 ARM_USART_GetRxCount 関数仕様

書式	uint32_t ARM_USART_GetRxCount(st_usart_resources_t const * const p_usart)
仕様説明	現時点の受信数を取得します
引数	st_usart_resources_t * const p_usart : USART のリソース 受信数を取得する USART のリソースを指定します。
戻り値	受信数

備考	インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	// 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();
	}

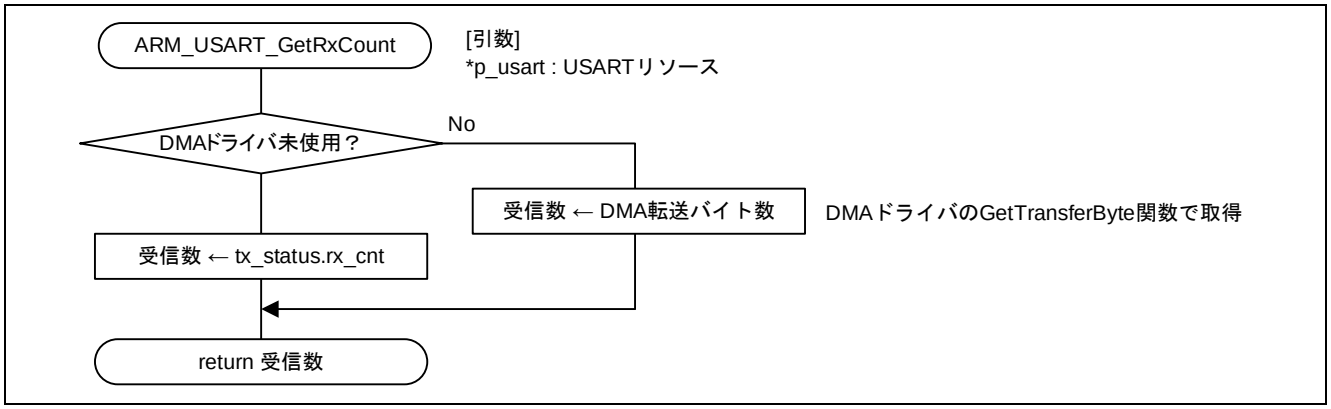


図 4-19 ARM_USART_GetRxCount 関数処理フロー

4.1.11 ARM_USART_Control 関数

表 4-11 ARM_USART_Control 関数仕様(1/2)

書式	int32_t ARM_USART_Control(uint32_t control, uint32_t arg, st_usart_resources_t const * const p_usart)
仕様説明	USART の制御コマンドを実行します
引数	uint32_t control : 制御コマンド 制御コマンドについては、「2.5.1 USART 制御コマンド定義」を参照
	uint32_t arg : コマンド別の引数（制御コマンドと引数の関係については表 4-13 参照）
	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	ARM_DRIVER_OK 制御コマンド実行成功
	ARM_DRIVER_ERROR 制御コマンド実行失敗 以下のいずれかの状態を検出すると制御コマンド実行失敗となります ・電源 OFF 状態で実行した場合 ・調歩同期モード設定(ARM_USART_MODE_ASYNCHRONOUS)で第 2 引数に 0 を指定した場合 ・クロック同期マスタモード設定(ARM_USART_MODE_SYNCHRONOUS_MASTER)で第 2 引数に 0 を指定した場合 ・RXI,ERI 割り込みを NVIC に登録していない状態でスマートカードモード(ARM_USART_MODE_SMART_CARD)を設定した場合 ・動作モード設定済みの状態で、再度動作モードを設定した場合 ・スマートカードクロック出力設定(ARM_USART_SET_SMART_CARD_CLOCK)でボーレート指定が異なる場合 ・スマートカードクロック出力設定をスマートカードモード以外で実行した場合 ・スマートカードクロック出力設定を送受信許可状態で実行した場合 ・スマートカード NACK 出力設定(ARM_USART_CONTROL_SMART_CARD_NACK)を許可設定以外で実行した場合 ・スマートカード NACK 出力設定をスマートカードモード以外で実行した場合 ・送信許可設定(ARM_USART_CONTROL_TX)を送信不可状態で実行した場合 ・受信許可設定(ARM_USART_CONTROL_RX)を受信不可状態で実行した場合 ・送受信許可設定(ARM_USART_CONTROL_TX_RX)を送信不可状態または受信不可状態で実行した場合 ・クロック同期モードにて送信許可状態で受信許可設定を実行した場合 ・クロック同期モードにて受信許可状態で送信許可設定を実行した場合 ・送信、または受信のどちらかのみ許可状態で送受信許可設定を実行した場合 ・動作モード設定前に送信許可設定、受信許可設定、送受信許可設定を実行した場合 ・送信禁止状態で送信中断設定(ARM_USART_ABORT_SEND)を実行した場合 ・受信禁止状態で受信中断設定(ARM_USART_ABORT_RECEIVE)を実行した場合 ・送信または受信禁止状態で送受信中断設定(ARM_USART_ABORT_TRANSFER)を実行した場合 ・不正なコマンドを実行した場合
	ARM_USART_ERROR_DATA_BITS データービット長設定エラー 調歩同期モード設定にてデータビット長に 7,8,9 ビット以外を設定した場合、データビット長設定エラーによる制御コマンド実行失敗となります

表 4-12 ARM_USART_Control 関数仕様(2/2)

戻り値	ARM_USART_ERROR_PARITY　　パリティ設定エラー 以下のいずれかの状態を検出するとパリティ設定エラーとなります ・調歩同期モード設定にてパリティなし、奇数パリティ、偶数パリティ以外を設定した場合 ・スマートカードモード設定にて奇数パリティ、偶数パリティ以外を設定した場合
	ARM_USART_ERROR_STOP_BITS　　ストップビット長設定エラー 調歩同期モード設定にてストップビット長に 1, 2 ビット以外を設定した場合、ストップビット長設定エラーとなります
	ARM_USART_ERROR_FLOW_CONTROL　フロー制御設定エラー 以下のいずれかの状態を検出するとフロー制御設定エラーとなります ・調歩同期モード設定にてフロー制御なし、CTS 制御、RTS 制御以外を設定した場合 ・クロック同期モード設定にてフロー制御なし、CTS 制御、RTS 制御以外を設定した場合
	ARM_USART_ERROR_BAUDRATE　　ボーレート設定エラー 指定したボーレートが実現不可の場合にボーレート設定エラーとなります
	ARM_USART_ERROR_MODE　　モード設定エラー 制御コマンドに ARM_USART_MODE_SINGLE_WIRE、または ARM_USART_MODE_IRDA を指定した場合、モード設定エラーとなります

備考	インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	　　sci0Drv->Control(ARM_USART_CONTROL_TX, 1);
	}

表 4-13 制御コマンドとコマンド別引数による動作

制御コマンド(control)	コマンド別引数(arg)	内容
ARM_USART_MODE_ASYNCHRONOUS	ボーレート （1～4294967295）	指定されたボーレートで調歩同期モードを初期化します
ARM_USART_MODE_SYNCHRONOUS_MASTER	ボーレート （1～4294967295）	指定されたボーレートでクロック同期マスタモードを初期化します
ARM_USART_MODE_SYNCHRONOUS_SLAVE	NULL(0)	引数は使用しません
ARM_USART_MODE_SMART_CARD	ボーレート （1～4294967295）	指定されたボーレートでスマートカードモードを初期化します
ARM_USART_SET_DEFAULT_TX_VALUE	デフォルトデータ値 (0x00～0xFF)	クロック同期モードの受信動作時に出力する送信データを設定します
ARM_USART_SET_SMART_CARD_CLOCK	出力ボーレート （1～4294967295）	スマートカードクロックの出力許可（出力ボーレートと現在のボーレートが一致していた場合のみ）
	0	スマートカードクロック出力禁止
ARM_USART_CONTROL_SMART_CARD_NACK	1	スマートカードモードの NACK 出力を許可にします（デフォルト許可状態）
ARM_USART_CONTROL_TX	1	送信を許可状態にします
	0	送信を禁止状態にします
ARM_USART_CONTROL_RX	1	受信を許可状態にします
	0	受信を禁止状態にします
ARM_USART_CONTROL_TX_RX	1	送受信を許可状態にします
	0	送受信を禁止状態にします
ARM_USART_ABORT_SEND	NULL(0)	引数は使用しません
ARM_USART_ABORT_RECEIVE	NULL(0)	引数は使用しません
ARM_USART_ABORT_TRANSFER	NULL(0)	引数は使用しません

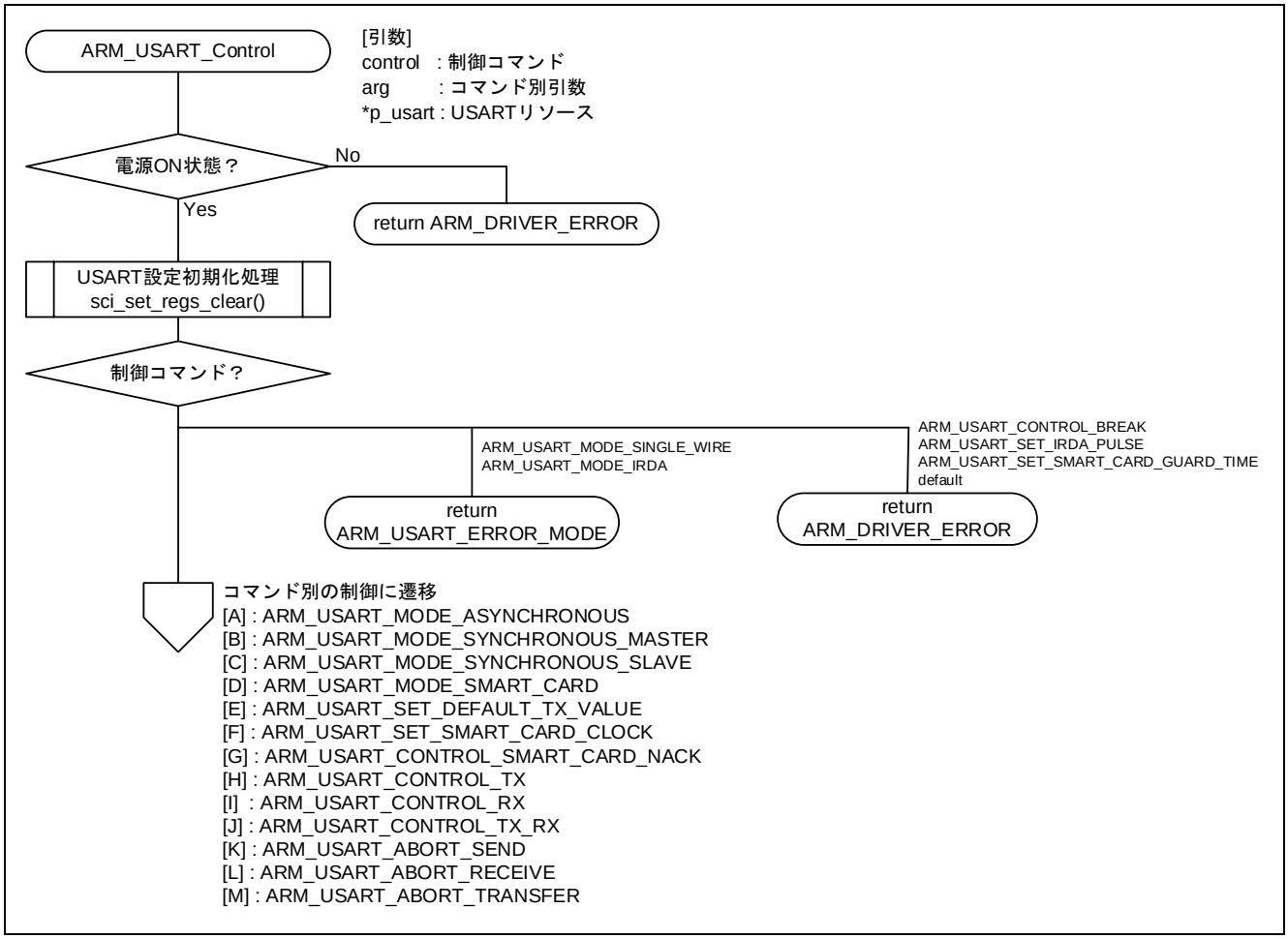


図 4-20 ARM_USART_Control 関数処理フロー(1/8)

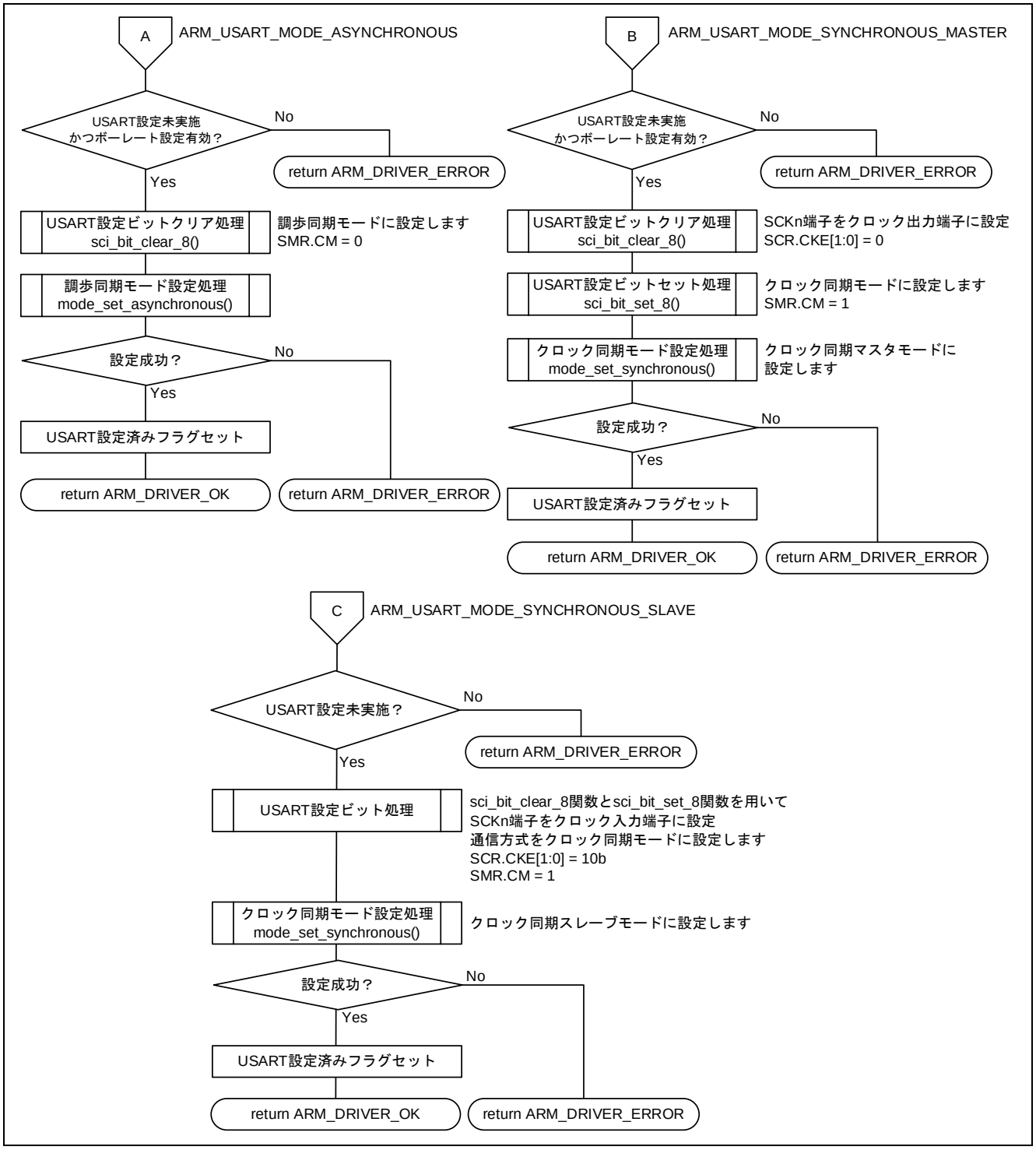


図 4-21 ARM_USART_Control 関数処理フロー(2/8)

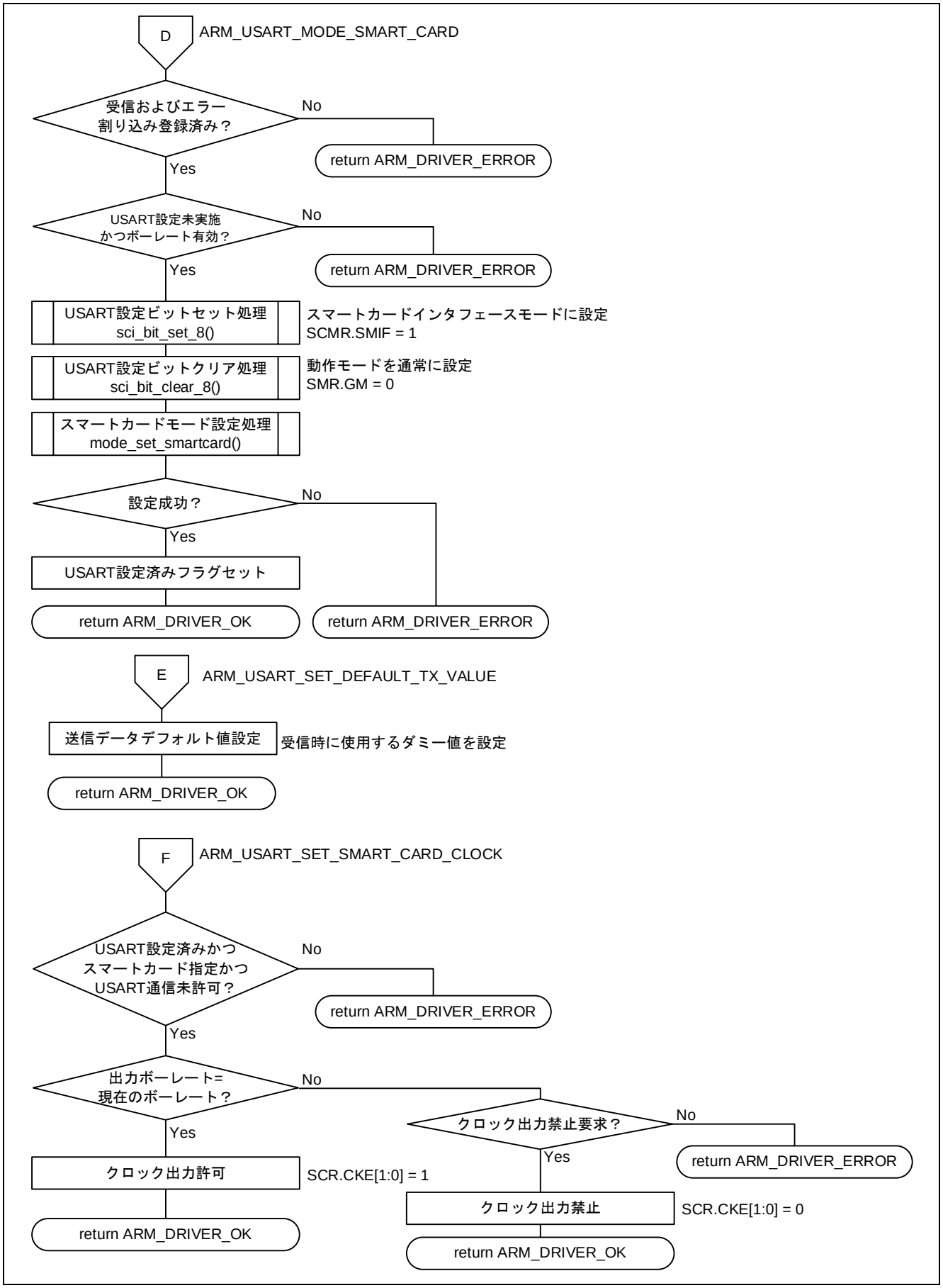


図 4-22 ARM_USART_Control 関数処理フロー(3/8)

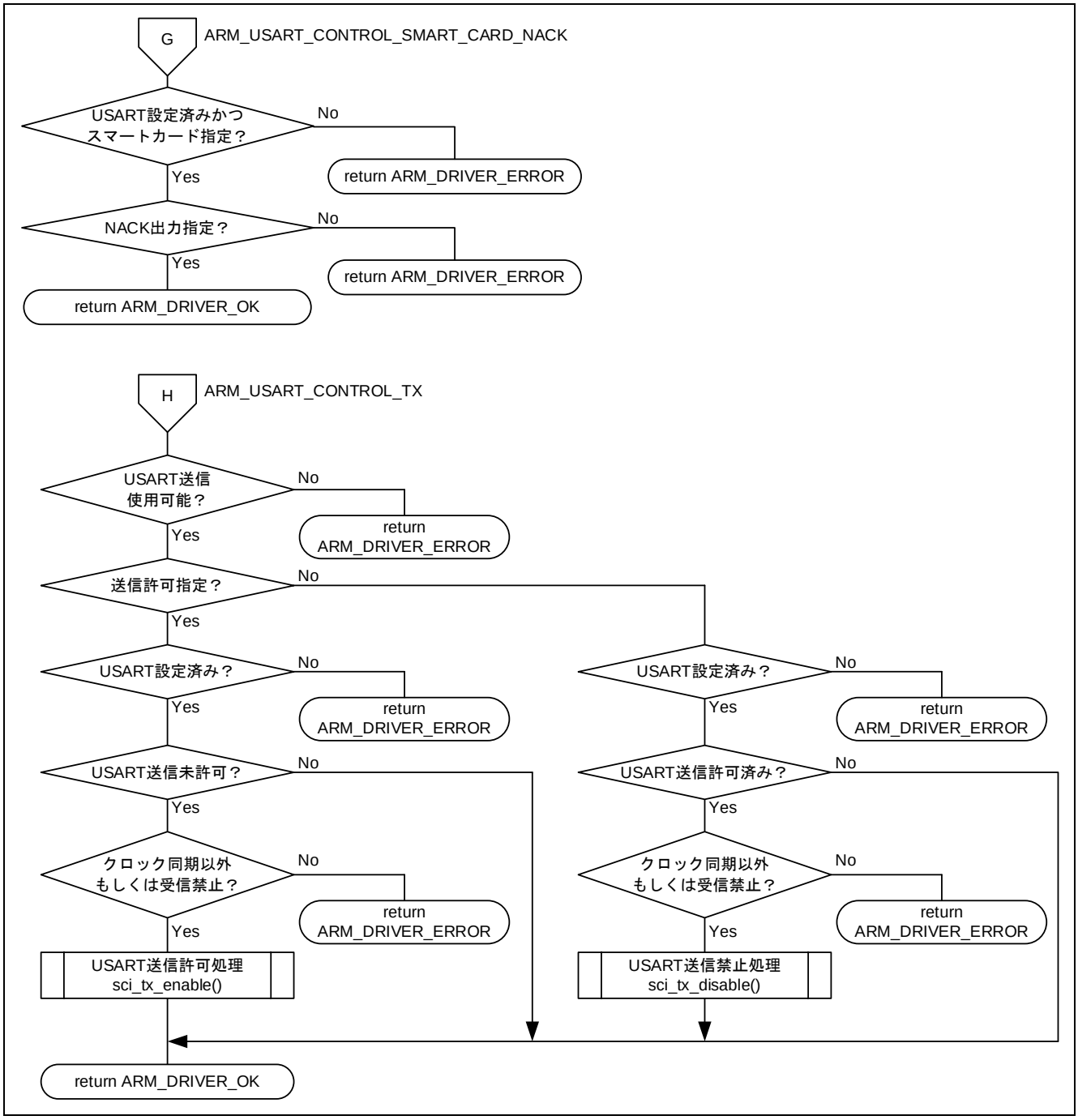


図 4-23 ARM_USART_Control 関数処理フロー(4/8)

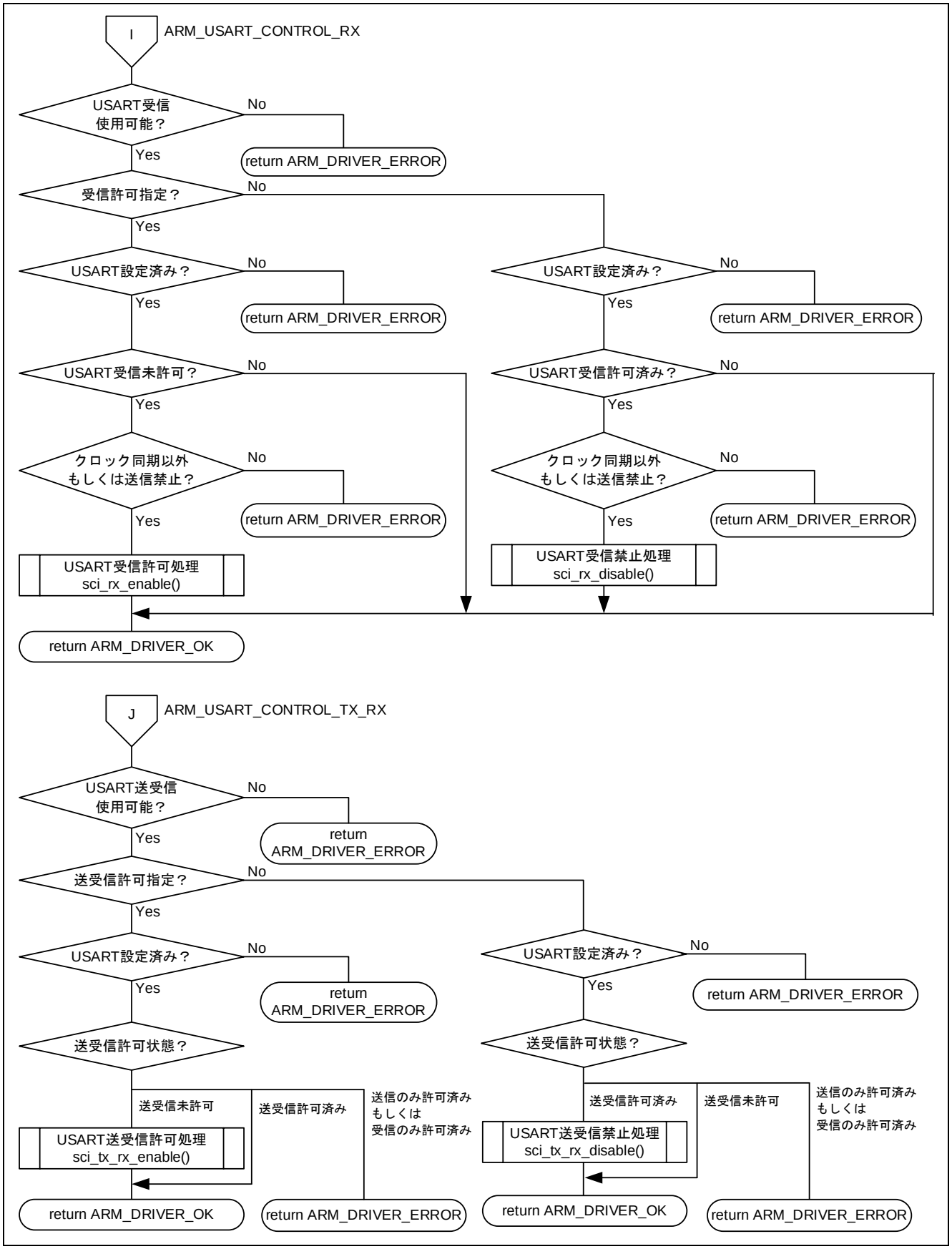


図 4-24 ARM_USART_Control 関数処理フロー(5/8)

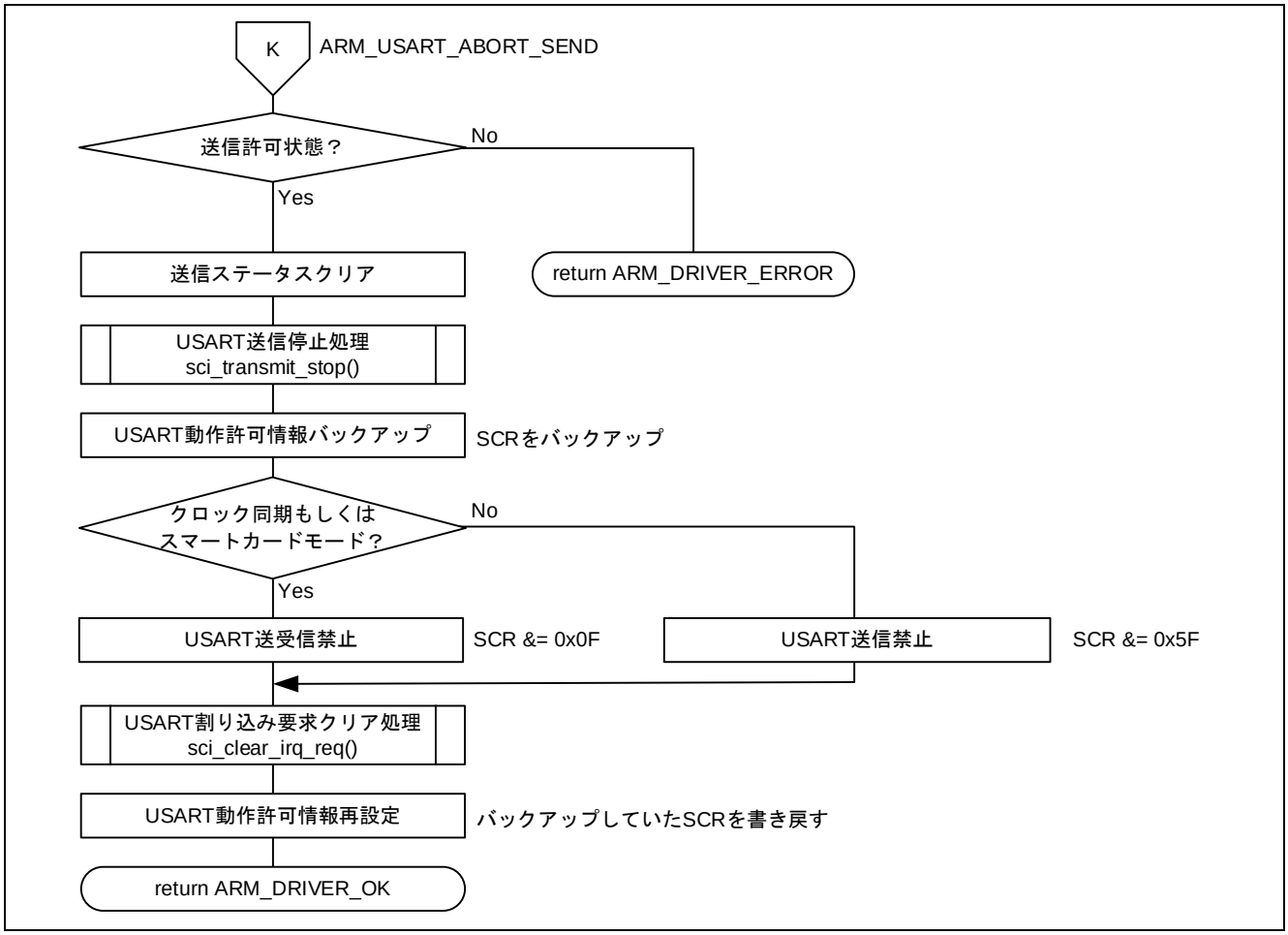


図 4-25 ARM_USART_Control 関数処理フロー(6/8)

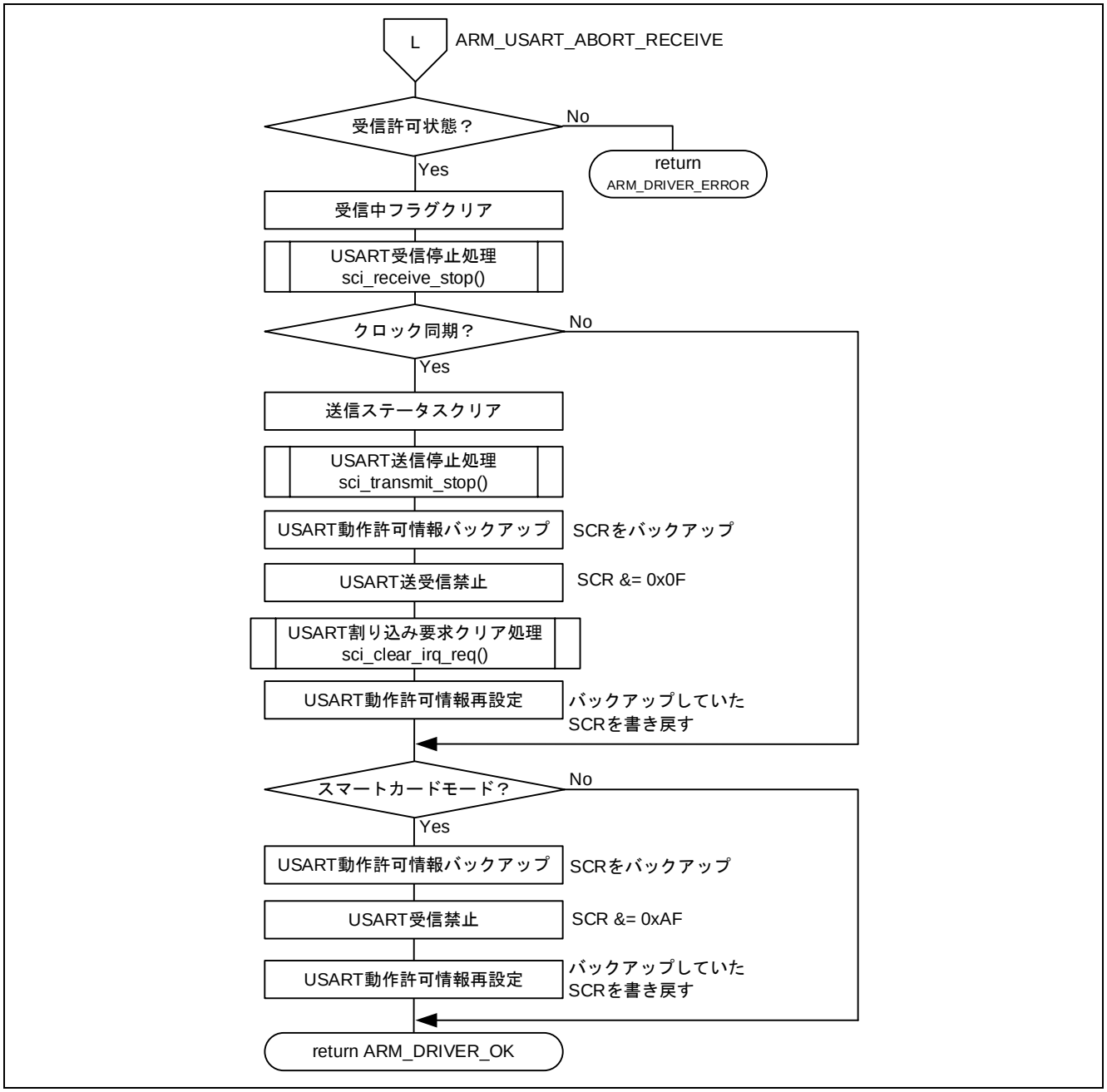


図 4-26 ARM_USART_Control 関数処理フロー(7/8)

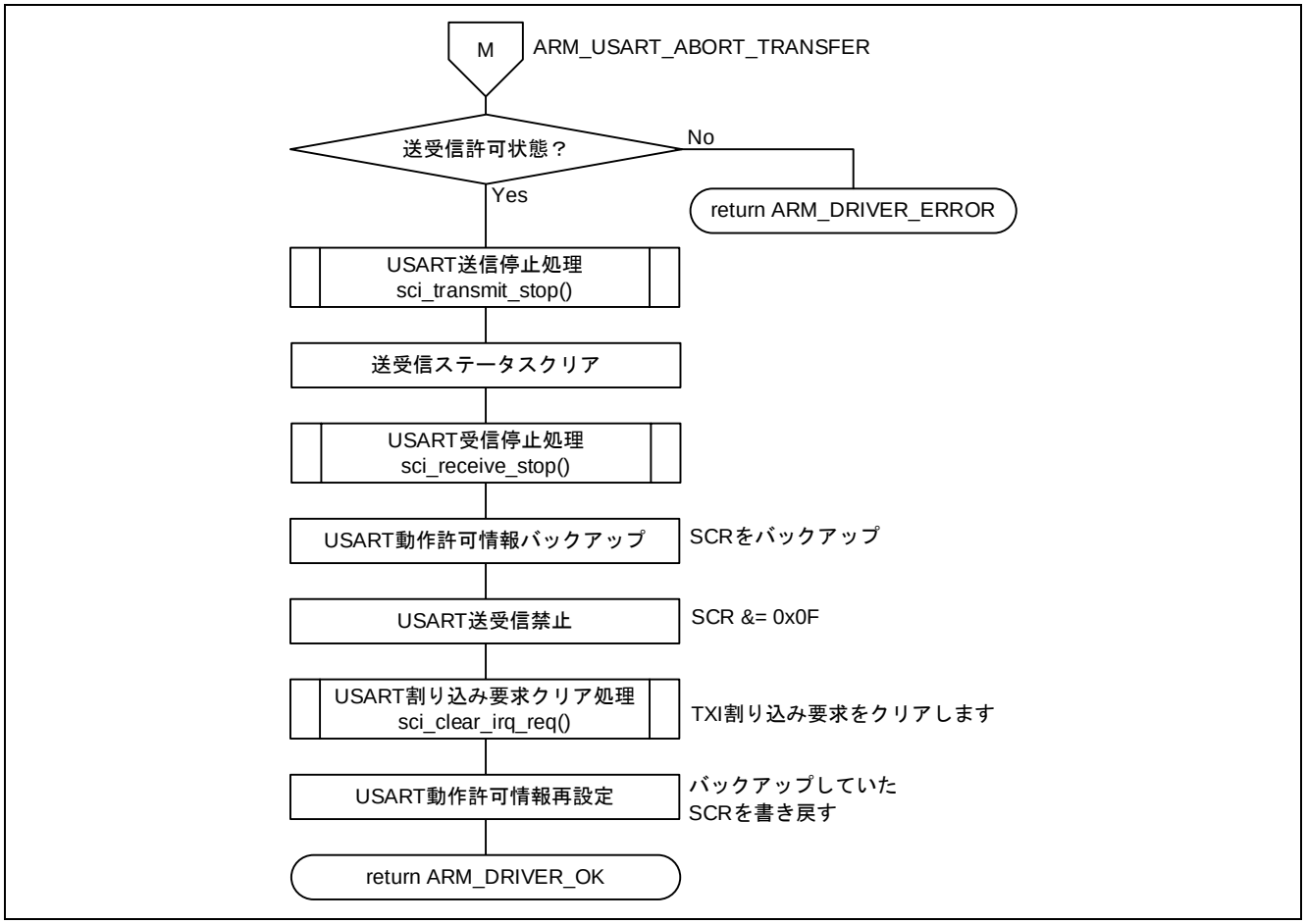


図 4-27 ARM_USART_Control 関数処理フロー(8/8)

4.1.12 ARM_USART_GetStatus 関数

表 4-14 ARM_USART_GetStatus 関数仕様

書式	ARM_USART_STATUS ARM_USART_GetStatus(st_usart_resources_t const * const p_usart)
仕様説明	USART のステータスを返します
引数	st_usart_resources_t * const p_usart : USART のリソース 対象の USART のリソースを指定します。
戻り値	通信ステータス

備考	インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	ARM_USART_STATUS state;
	state = sci0Drv->GetStatus();
	
	}

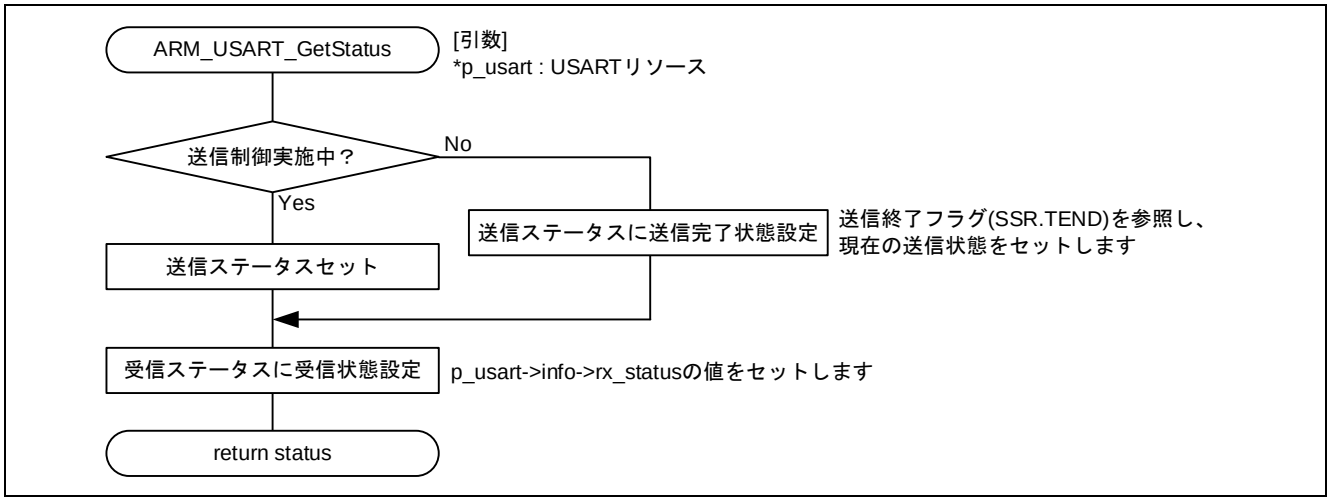


図 4-28 ARM_USART_GetStatus 関数処理フロー

4.1.13 ARM_USART_SetModemControl 関数

表 4-15 ARM_USART_SetModemControl 関数仕様

書式	int32_t ARM_USART_SetModemControl(ARM_USART_MODEM_CONTROL control, st_usart_resources_t const * const p_usart)
仕様説明	ソフトウェアによるモデム制御を行います
引数	ARM_USART_MODEM_CONTROL control : モデム制御コマンド モデム制御コマンドについては「2.5.3 モデム制御定義 ARM_USART_MODEM_STATUS 構造体」を参照
	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	ARM_DRIVER_OK モデム制御成功
	ARM_DRIVER_ERROR モデム制御失敗 以下のいずれかの状態を検出すると送受信開始失敗となります ・r_usart_cfg.h にて RTS 端子設定を設定せず実行した場合 ・USART 設定実施前に実行した場合 ・RTS 制御設定で動作中に実行した場合 ・モデム制御コマンドに不正なコマンドを設定した場合

備考	インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	// USART driver instance (SCI0)
	extern ARM_DRIVER_USART Driver_USART0;
	ARM_DRIVER_USART *sci0Drv = &Driver_USART0;
	
	main()
	{
	sci0Drv->SetModemControl(ARM_USART_RTS_SET);
	
	}

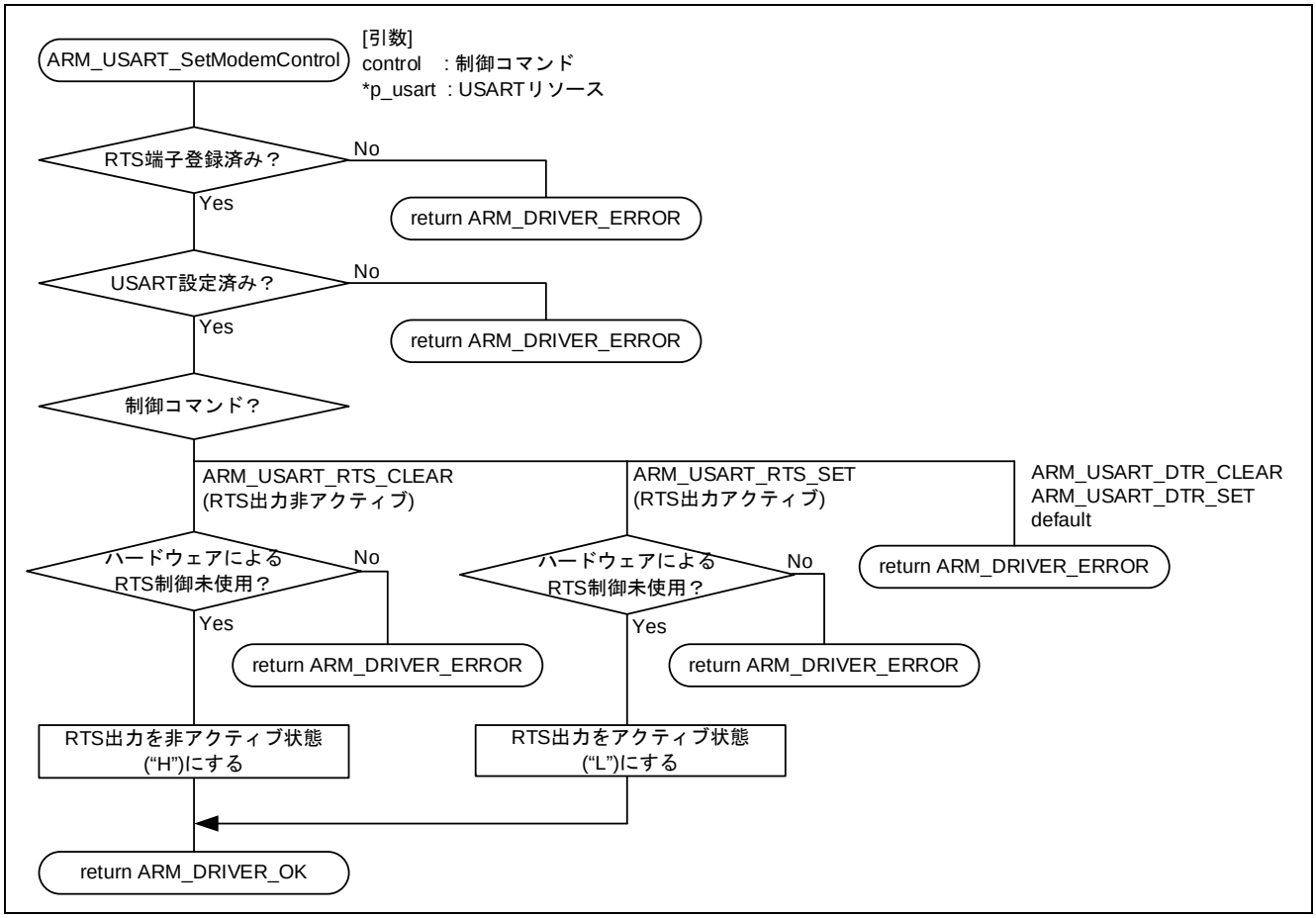


図 4-29 ARM_USART_SetModemControl 関数処理フロー

4.1.14 ARM_USART_GetModemStatus 関数

表 4-16 ARM_USART_GetModemStatus 関数仕様

書式	ARM_USART_MODEM_STATUS ARM_USART_GetModemStatus(st_usart_resources_t const * const p_usart)
仕様説明	CTS 端子の状態を取得します
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	モデム状態

備考	本関数はハードウェアによる CTS 制御未使用時のみ有効です。
	また、インスタンスからのアクセス時は USART リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	// 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();
	
	}

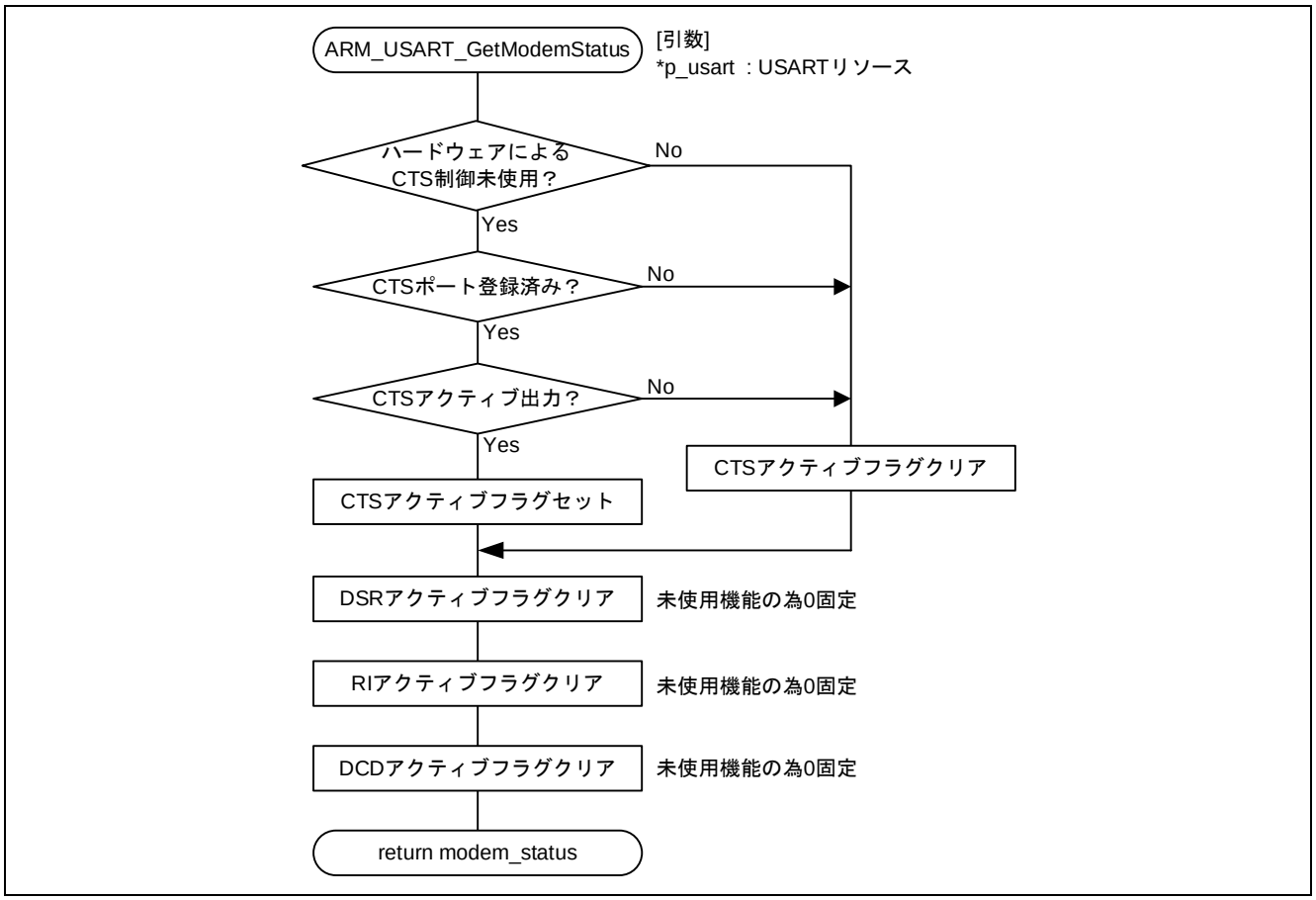


図 4-30 ARM_USART_GetModemStatus 関数処理フロー

4.1.15 mode_set_asynchronous 関数

表 4-17 mode_set_asynchronous 関数仕様

書式	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)
仕様説明	調歩同期モードの設定を行います
引数	uint32_t control : 制御コマンド
	st_sci_reg_set_t * const p_sci_regs : レジスタ設定値格納ポインタ
	uint32_t baud : ボーレート設定
	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	ARM_DRIVER_OK 調歩同期設定成功
	ARM_USART_ERROR_DATA_BITS データビット長設定エラー データビット長に 7,8,9 ビット以外を設定した場合、データビット長設定エラーとなります
	ARM_USART_ERROR_PARITY パリティ設定エラー パリティなし、奇数パリティ、偶数パリティ以外を設定した場合、パリティ設定エラーとなります
	ARM_USART_ERROR_STOP_BITS ストップビット長設定エラー ストップビット長に 1, 2 ビット以外を設定した場合、ストップビット長設定エラーとなります
	ARM_USART_ERROR_FLOW_CONTROL フロー制御設定エラー フロー制御なし、CTS 制御、RTS 制御以外を設定した場合、フロー制御設定エラーとなります
	ARM_USART_ERROR_BAUDRATE ボーレート設定エラー 指定したボーレートが実現不可の場合にボーレート設定エラーとなります
備考	―

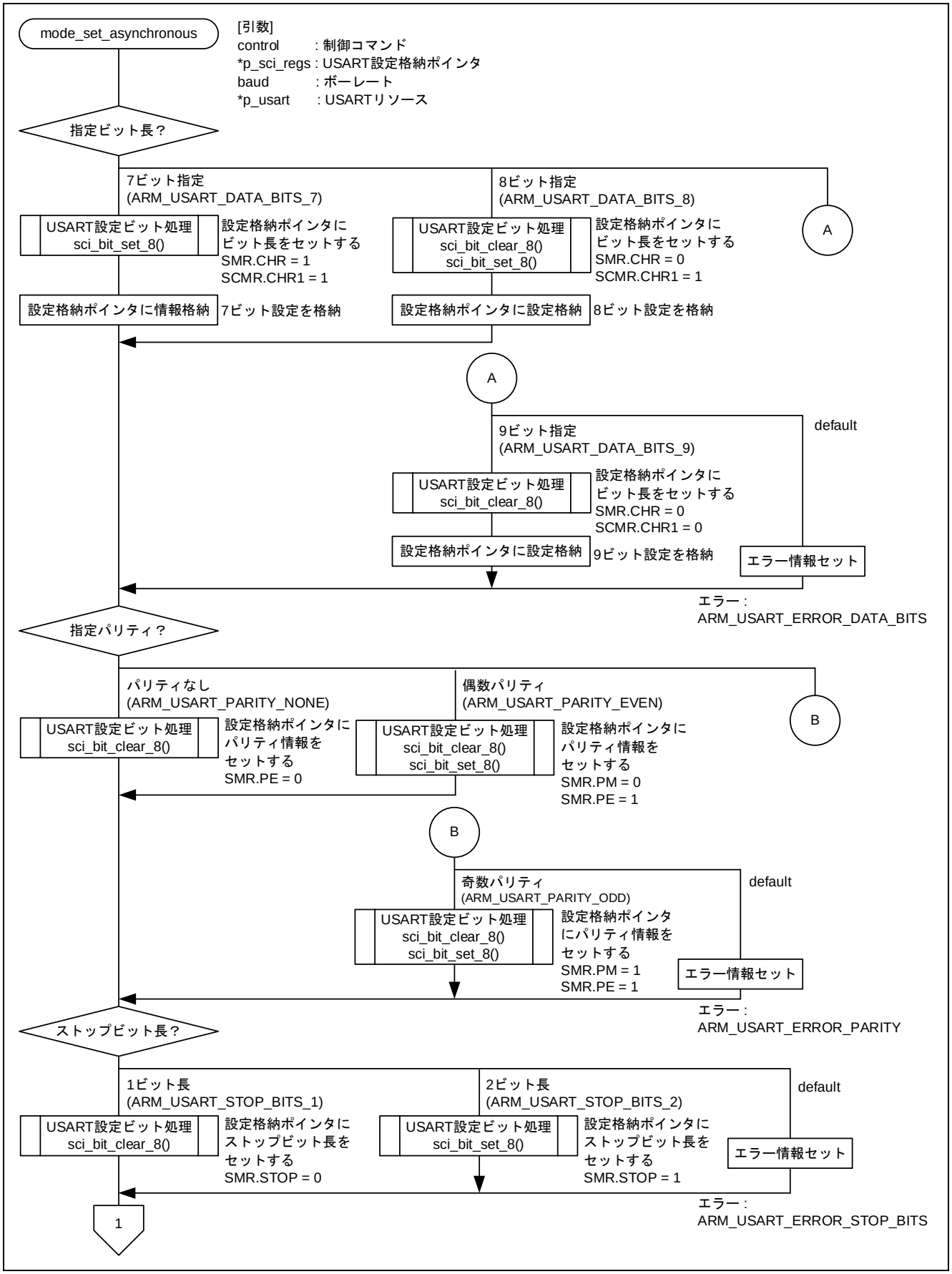


図 4-31 mode_set_asynchronous 関数処理フロー(1/2)

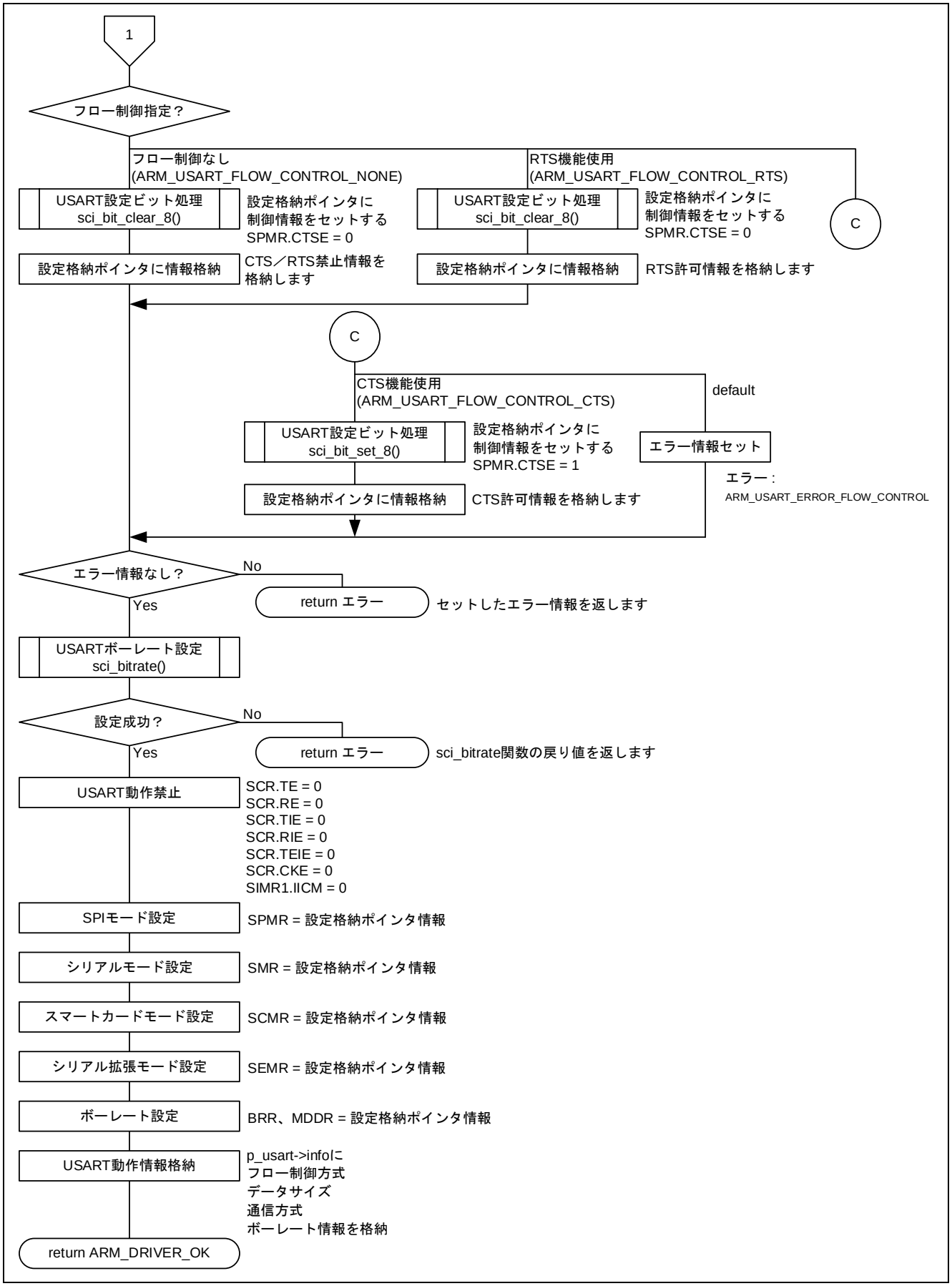


図 4-32 mode_set_asynchronous 関数処理フロー(2/2)

4.1.16 mode_set_synchronous 関数

表 4-18 mode_set_synchronous 関数仕様

書式	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)
仕様説明	クロック同期モードの設定を行います
引数	uint32_t control : 制御コマンド
	st_sci_reg_set_t * const p_sci_regs : レジスタ設定値格納ポインタ
	uint32_t baud : ボーレート設定
	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	ARM_DRIVER_OK クロック同期設定成功
	ARM_USART_ERROR_FLOW_CONTROL フロー制御設定エラー フロー制御なし、CTS 制御、RTS 制御以外を設定した場合、フロー制御設定エラーとなります
	ARM_USART_ERROR_BAUDRATE ボーレート設定エラー 指定したボーレートが実現不可の場合にボーレート設定エラーとなります
備考	－

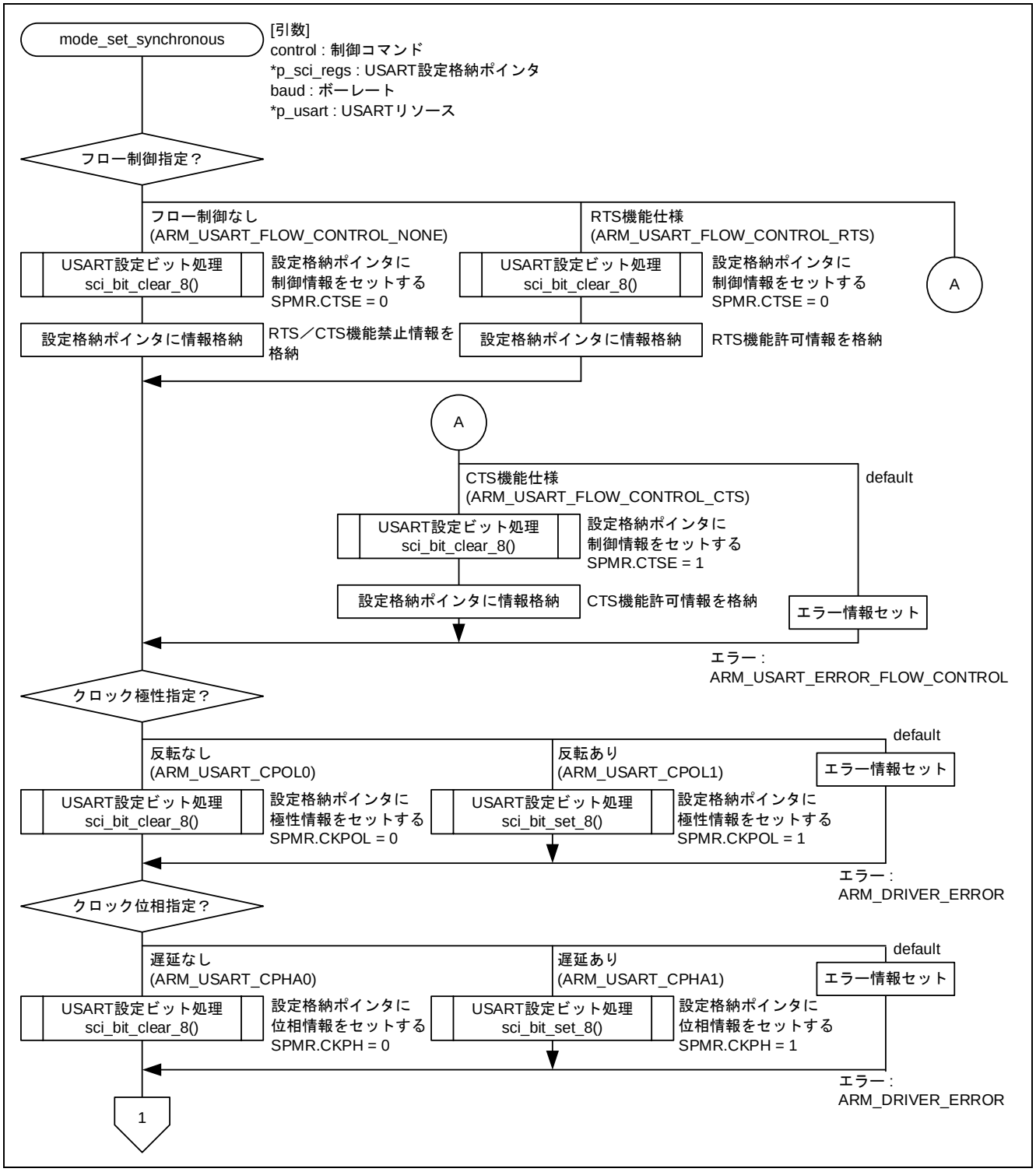


図 4-33 mode_set_synchronous 関数処理フロー(1/2)

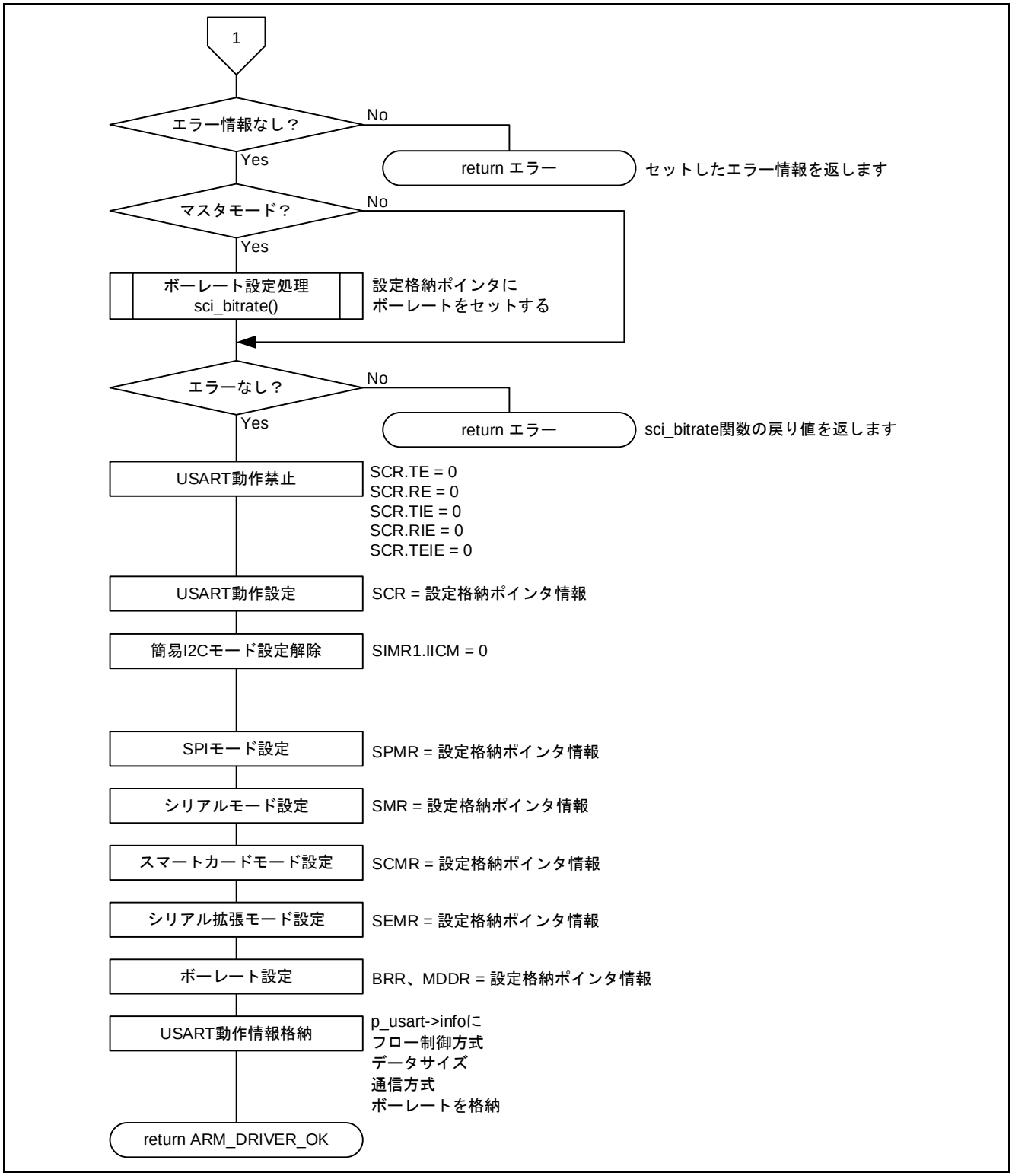


図 4-34 mode_set_synchronous 関数処理フロー(2/2)

4.1.17 mode_set_smartcard 関数

表 4-19 mode_set_smartcard 関数仕様

書式	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)
仕様説明	スマートカードモードの設定を行います
引数	uint32_t control : 制御コマンド
	st_sci_reg_set_t * const p_sci_regs : レジスタ設定値格納ポインタ
	uint32_t baud : ボーレート設定
	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	ARM_DRIVER_OK　スマートカード設定成功
	ARM_USART_ERROR_PARITY　パリティ設定エラー 奇数パリティ、偶数パリティ以外を設定した場合、パリティ設定エラーとなります
	ARM_USART_ERROR_BAUDRATE　ボーレート設定エラー 指定したボーレートが実現不可の場合にボーレート設定エラーとなります
備考	－

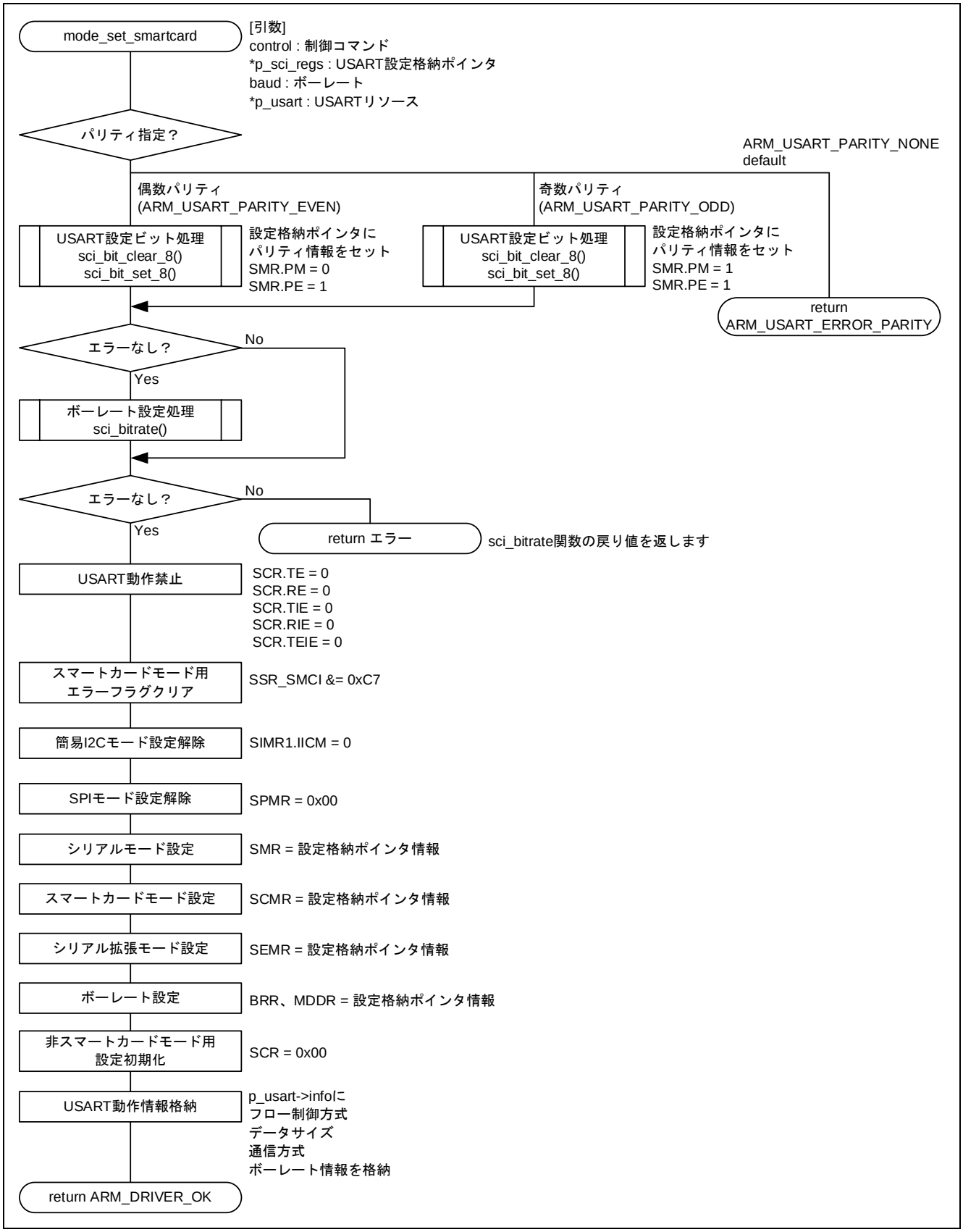
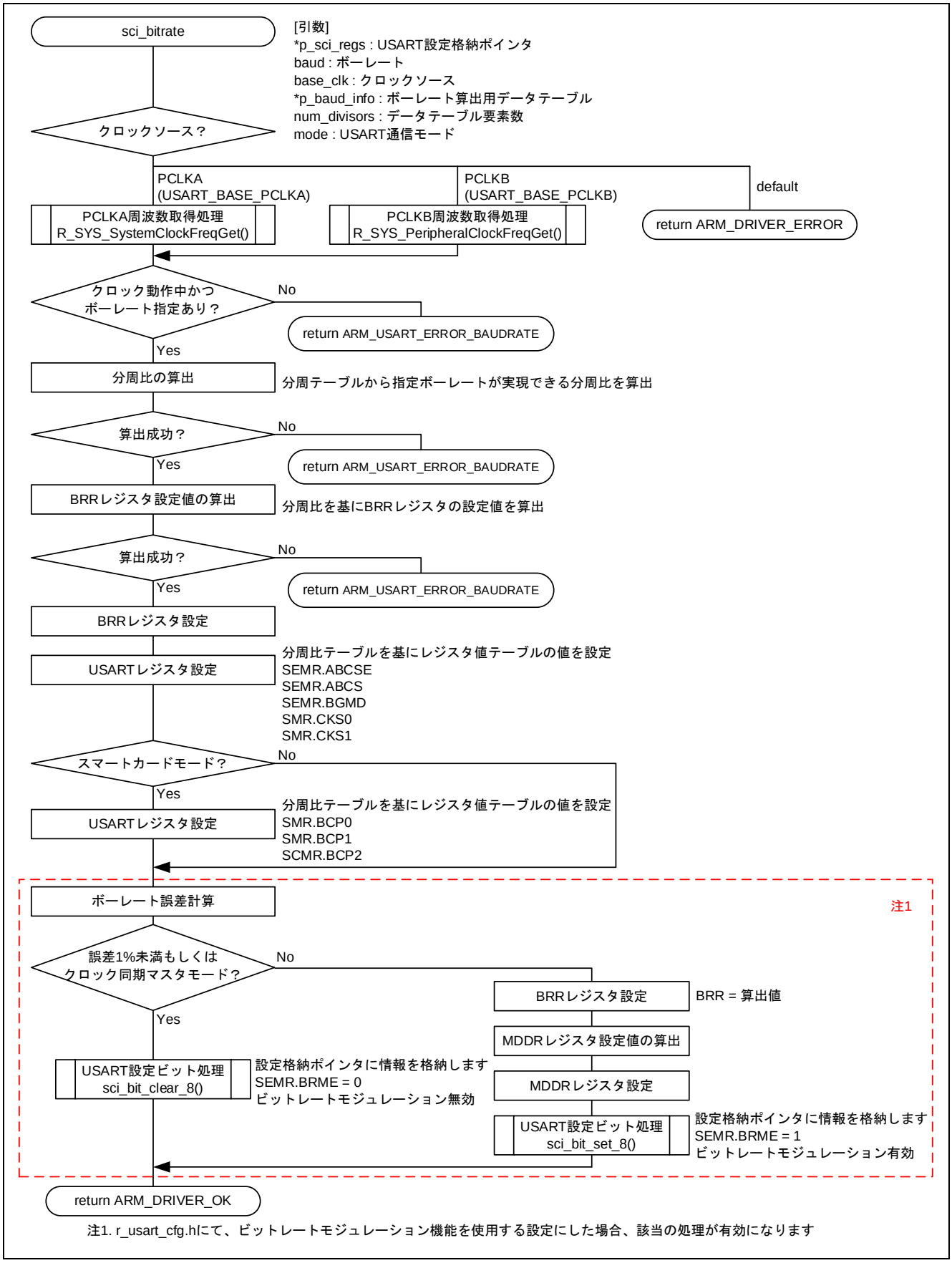


図 4-35 mode_set_smartcard 関数処理フロー

4.1.18 sci_bitrate 関数

表 4-20 sci_bitrate 関数仕様

書式	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)
仕様説明	ボーレートの算出を行います
引数	st_sci_reg_set_t *p_sci_regs : レジスタ設定用バッファポインタ バス速度の算出結果を格納するバッファポインタ
	uint32_t baud : バス速度
	e_usart_base_clk_t base_clk : ベースクロック
	st_baud_divisor_t const *p_baud_info : 分周比テーブル
	uint8_t num_divisors : 分周テーブルサイズ
	uint32_t mode : 動作モード
戻り値	ARM_DRIVER_OK　　バス速度算出成功
	ARM_DRIVER_ERROR　　バス速度算出エラー ベースクロックが PCLKA または PCLKB 以外を選択した場合、バス速度算出エラーとなります
	ARM_USART_ERROR_BAUDRATE　　ボーレート算出失敗
備考	―



注1. r_usart_cfg.hにて、ビットレートモジュレーション機能を使用する設定にした場合、該当の処理が有効になります

図 4-36 sci_bitrate 関数処理フロー

4.1.19 sci_set_regs_clear 関数

表 4-21 sci_set_regs_clear 関数仕様

書式	static void sci_set_regs_clear(st_sci_reg_set_t * const p_regs)
仕様説明	レジスタ設定用バッファの初期化を実施します
引数	st_sci_reg_set_t *p_sci_regs : レジスタ設定用バッファポインタ
戻り値	なし
備考	―

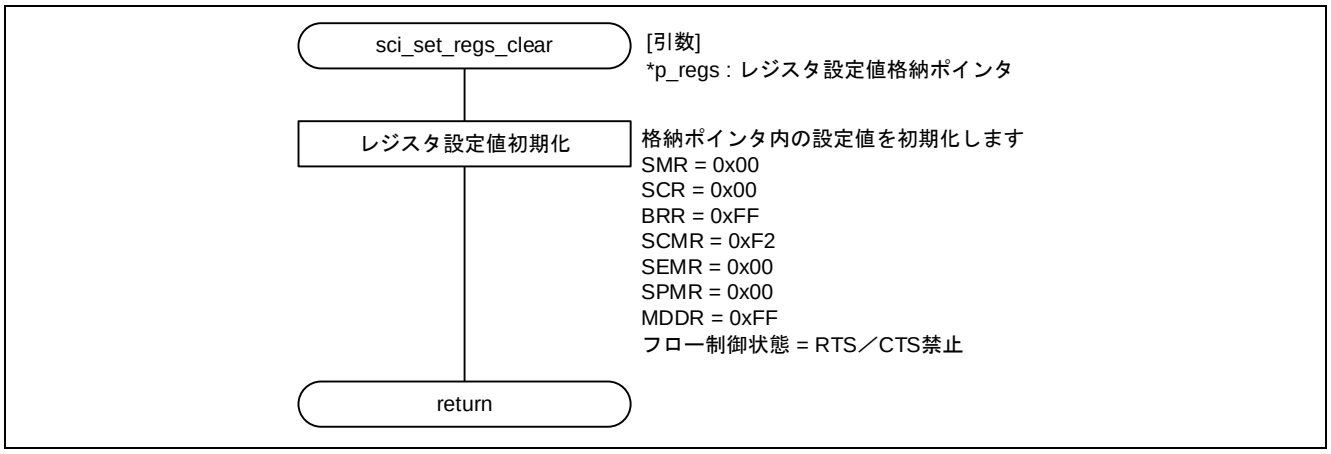


図 4-37 sci_set_regs_clear 関数処理フロー

4.1.20 sci_tx_enable 関数

表 4-22 sci_tx_enable 関数仕様

書式	static void sci_tx_enable(st_usart_resources_t * const p_usart)
仕様説明	送信許可設定を行います
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	―

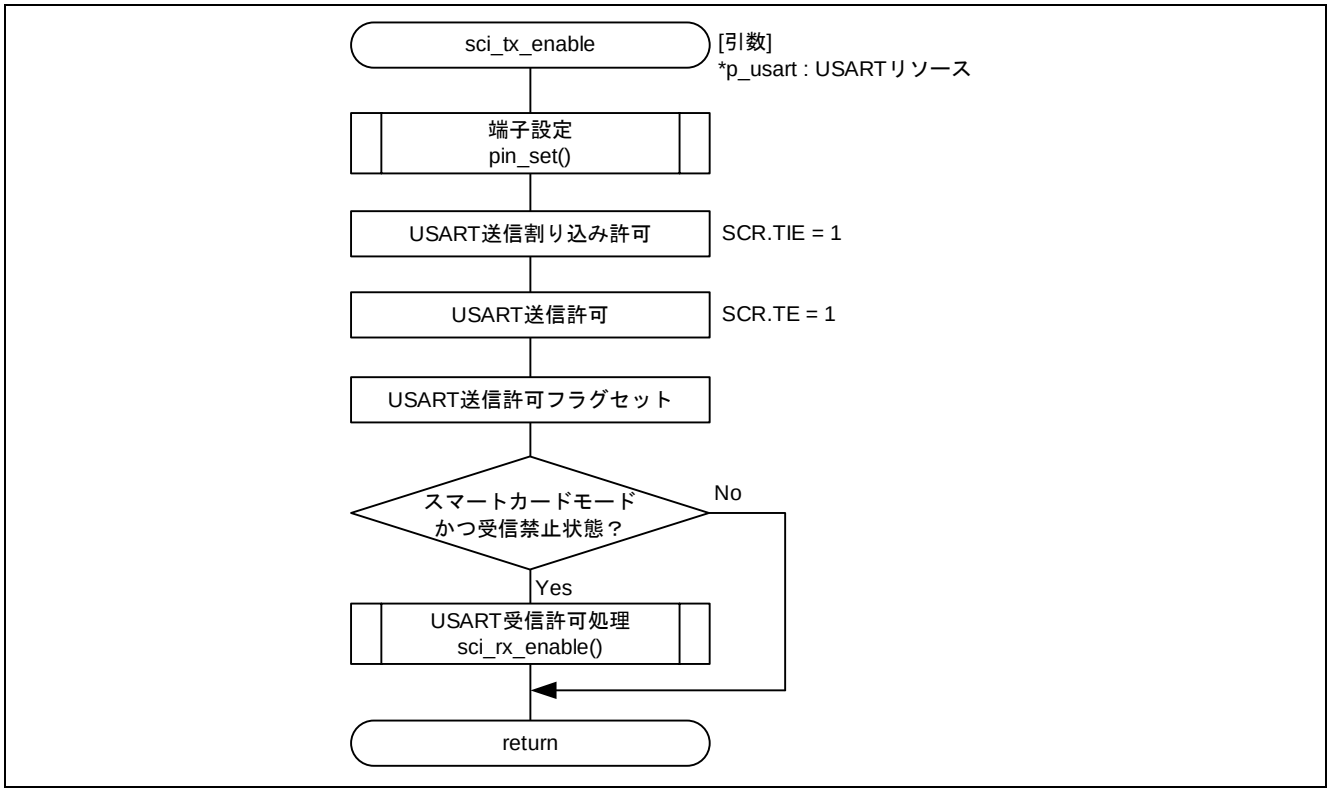


図 4-38 sci_tx_enable 関数処理フロー

4.1.21 sci_tx_disable 関数

表 4-23 sci_tx_disable 関数仕様

書式	static void sci_tx_disable(st_usart_resources_t * const p_usart)
仕様説明	送信禁止設定を行います
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	―

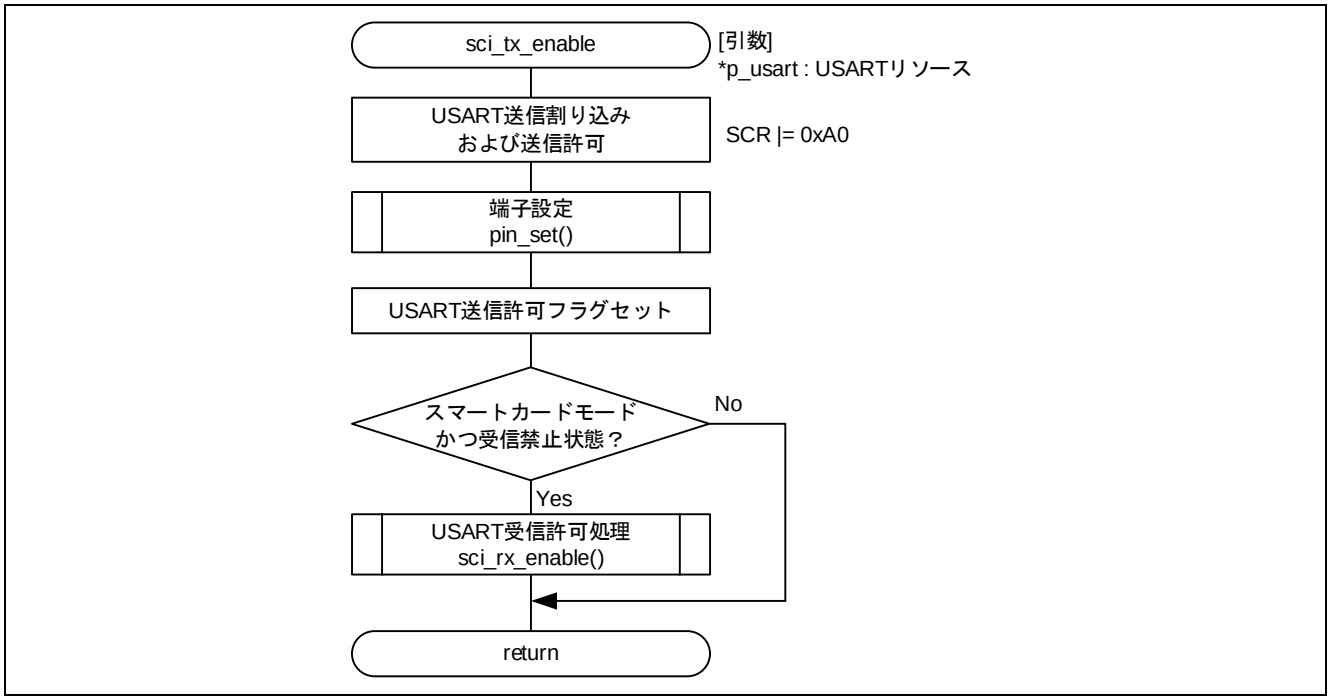


図 4-39 sci_tx_disable 関数処理フロー

4.1.22 sci_rx_enable 関数

表 4-24 sci_rx_enable 関数仕様

書式	static void sci_rx_enable(st_usart_resources_t * const p_usart)
仕様説明	受信許可設定を行います
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	―

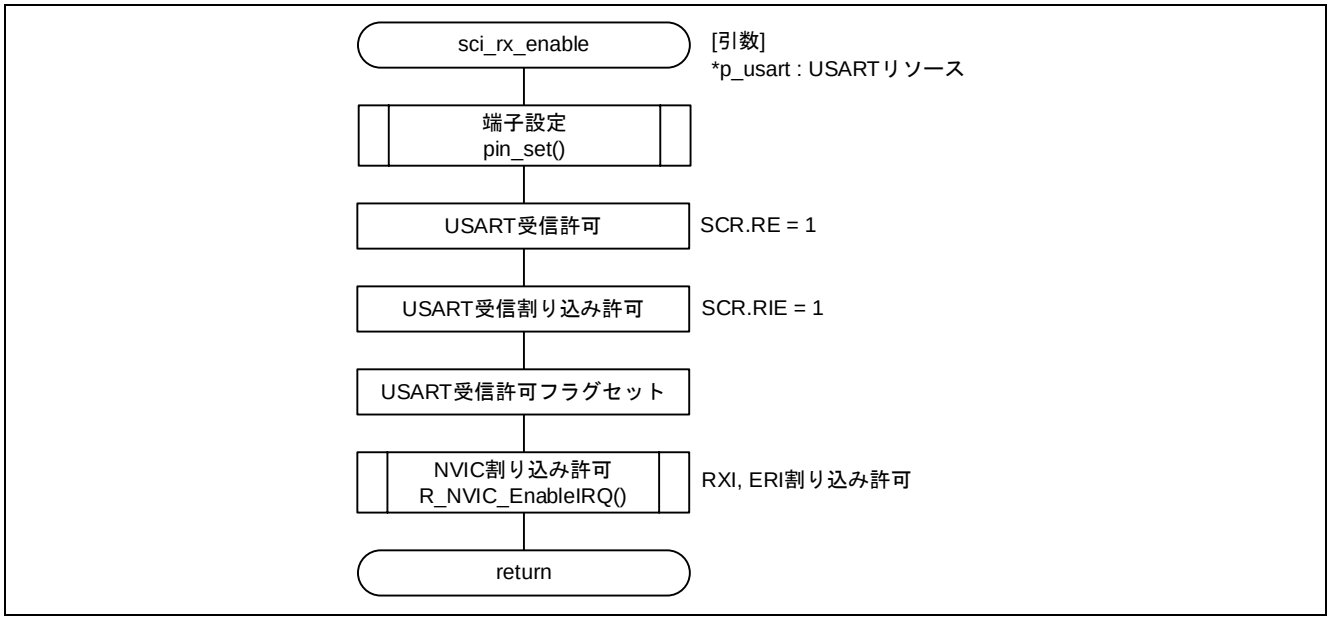


図 4-40 sci_rx_enable 関数処理フロー

4.1.23 sci_rx_disable 関数

表 4-25 sci_rx_disable 関数仕様

書式	static void sci_rx_disable(st_usart_resources_t * const p_usart)
仕様説明	受信禁止設定を行います
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	―

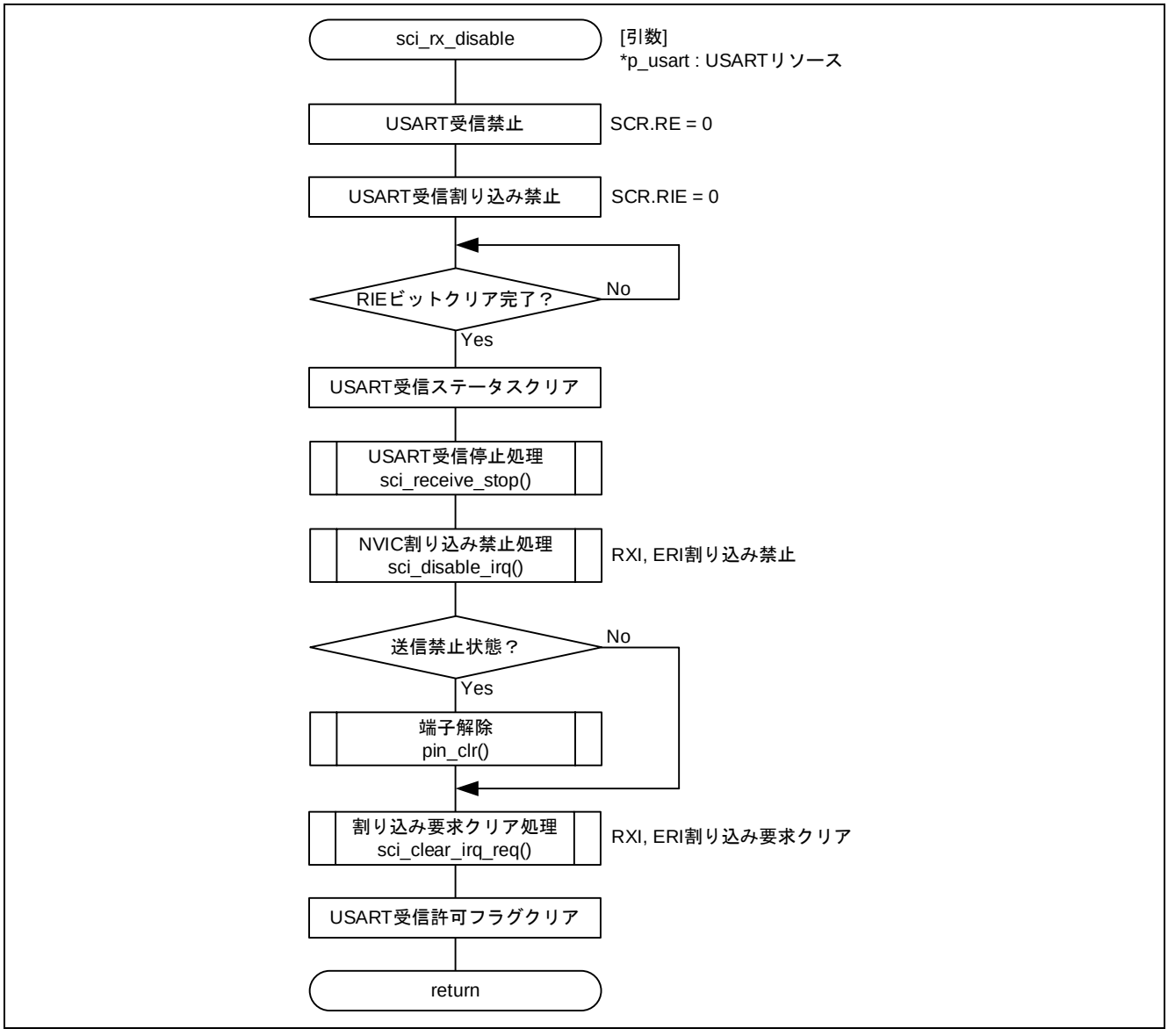


図 4-41 sci_rx_disable 関数処理フロー

4.1.24 sci_tx_rx_enable 関数

表 4-26 sci_tx_rx_enable 関数仕様

書式	static void sci_tx_rx_enable (st_usart_resources_t * const p_usart)
仕様説明	送受信許可設定を行います
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	―

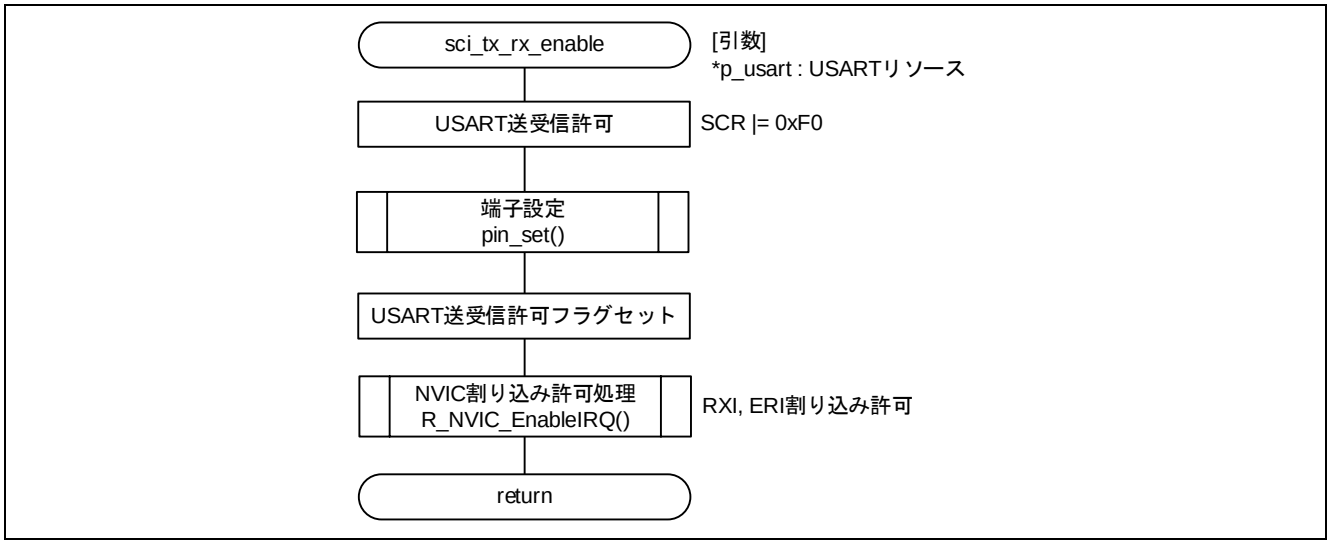


図 4-42 sci_tx_rx_enable 関数処理フロー

4.1.25 sci_tx_rx_disable 関数

表 4-27 sci_tx_rx_disable 関数仕様

書式	static void sci_tx_rx_disable (st_usart_resources_t * const p_usart)
仕様説明	送受信禁止設定を行います
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	―

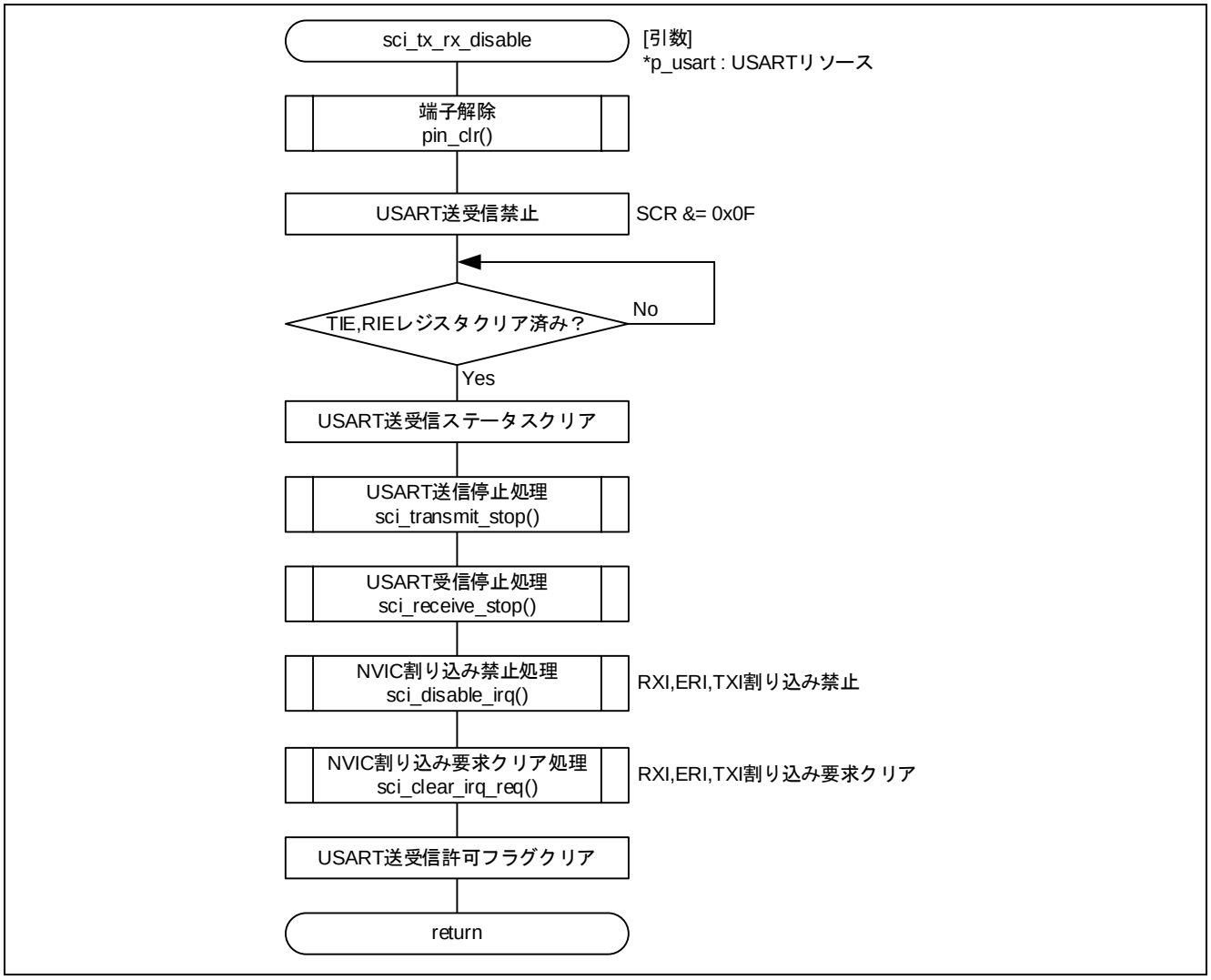


図 4-43 sci_tx_rx_disable 関数処理フロー

4.1.26 check_tx_available 関数

表 4-28 check_tx_available 関数仕様

書式	static int32_t check_tx_available(int16_t * const p_flag, st_usart_resources_t * const p_usart)
仕様説明	送信可能かどうかの判定を行います
引数	int16_t * const p_flag: 初期化フラグ格納ポインタ
	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	ARM_DRIVER_OK 送信可否判定成功
	ARM_DRIVER_ERROR 送信可否判定失敗 以下のいずれかの状態を検出すると送信可否判定失敗となります ・送信処理に割り込み、または DTC を使用時、TXI 割り込みのイベントリンク設定に失敗した場合 ・送信処理に割り込み、または DTC を使用時、割り込み優先レベルの設定に失敗した場合 ・送信処理に DTC を使用時、r_system_cfg.h で TXI 割り込みが未使用定義（SYSTEM_IRQ_EVENT_NUMBER_NOT_USED）になっている場合 ・送信処理に DTC、または DMAC を使用時、DMA ドライバの初期化に失敗した場合 ・送信処理に DMAC を使用時、DMAC 割り込み許可設定に失敗した場合
備考	ー

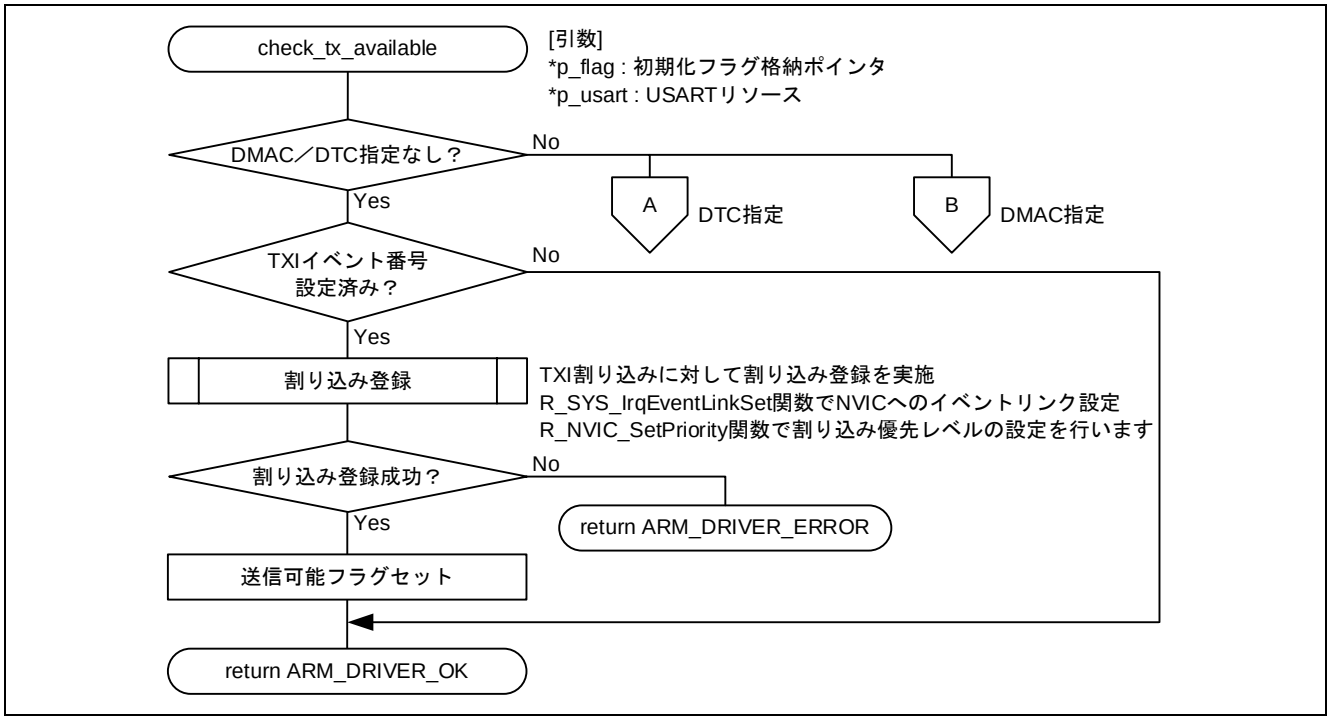
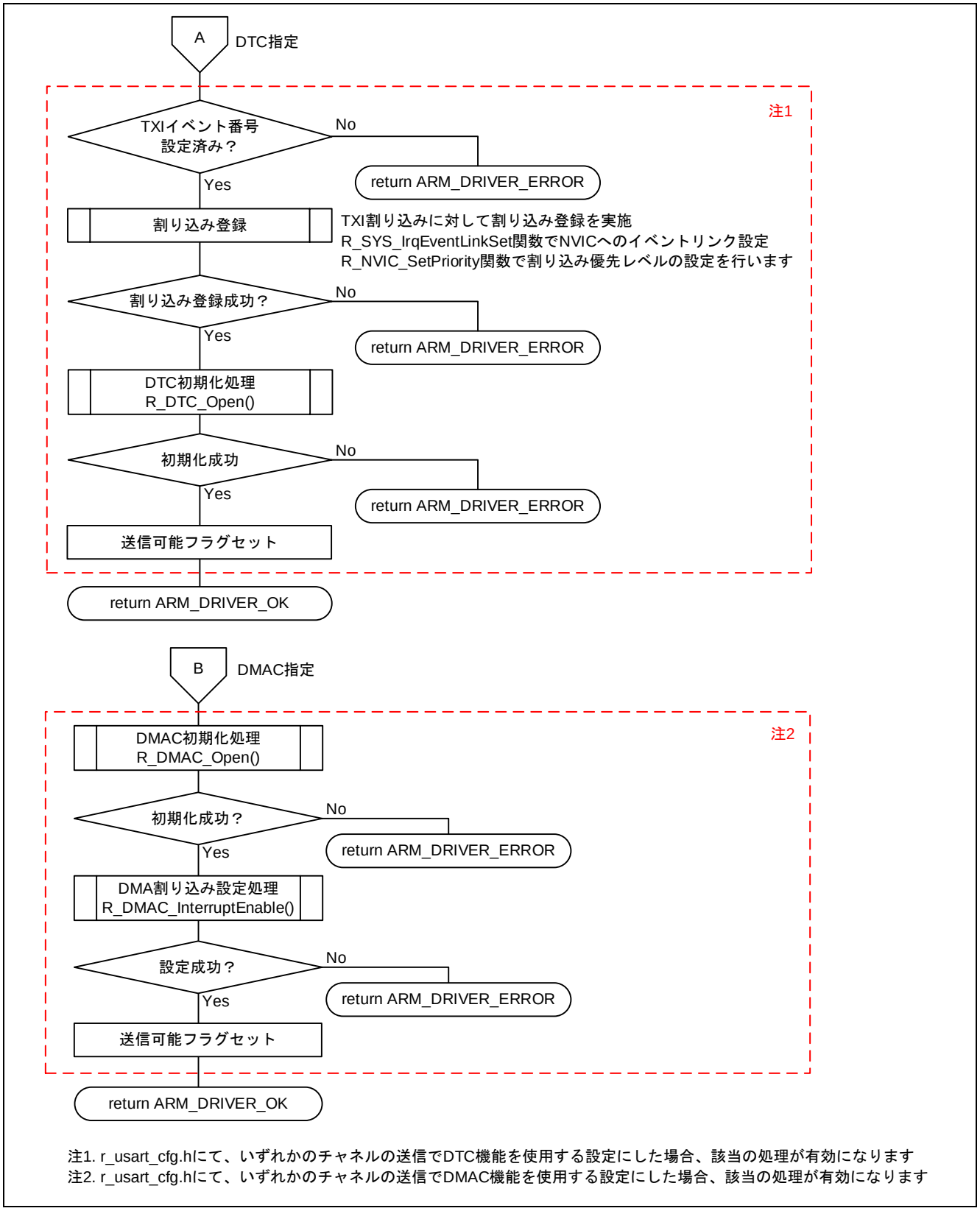


図 4-44 check_tx_available 関数処理フロー(1/2)



注1. r_usart_cfg.hにて、いずれかのチャネルの送信でDTC機能を使用する設定にした場合、該当の処理が有効になります
注2. r_usart_cfg.hにて、いずれかのチャネルの送信でDMAC機能を使用する設定にした場合、該当の処理が有効になります

図 4-45 check_tx_available 関数処理フロー(2/2)

4.1.27 check_rx_available 関数

表 4-29 check_rx_available 関数仕様

書式	static int32_t check_rx_available(int16_t * const p_flag, st_usart_resources_t * const p_usart)
仕様説明	受信可能かどうかの判定を行います
引数	int16_t * const p_flag: 初期化フラグ格納ポインタ
	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	ARM_DRIVER_OK 受信可否判定成功
	ARM_DRIVER_ERROR 受信可否判定失敗 以下のいずれかの状態を検出すると受信可否判定失敗となります ・RXI、ERI 割り込みのイベントリンク設定に失敗した場合 ・RXI、ERI 割り込みの割り込み優先レベルの設定に失敗した場合 ・受信処理に DTC を使用時、r_system_cfg.h で RXI、ERI 割り込みが未使用定義（SYSTEM_IRQ_EVENT_NUMBER_NOT_USED）になっている場合 ・受信処理に DTC、または DMAC を使用時、DMA ドライバの初期化に失敗した場合 ・受信処理に DMAC を使用時、DMAC 割り込み許可設定に失敗した場合
備考	ー

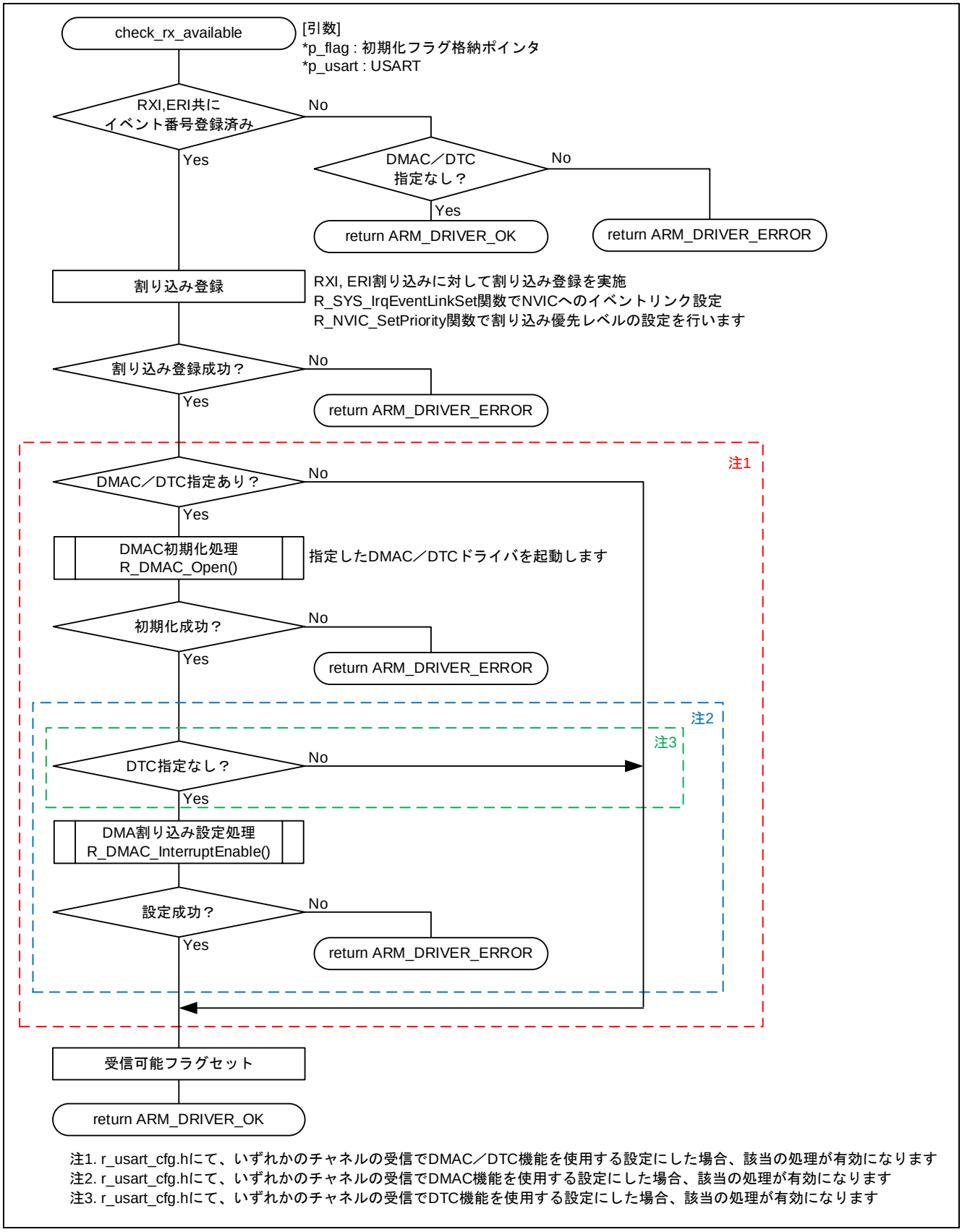


図 4-46 check_rx_available 関数処理フロー

4.1.28 sci_transmit_stop 関数

表 4-30 sci_transmit_stop 関数仕様

書式	static void sci_transmit_stop(st_usart_resources_t const * const p_usart)
仕様説明	送信を中断します
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	ー

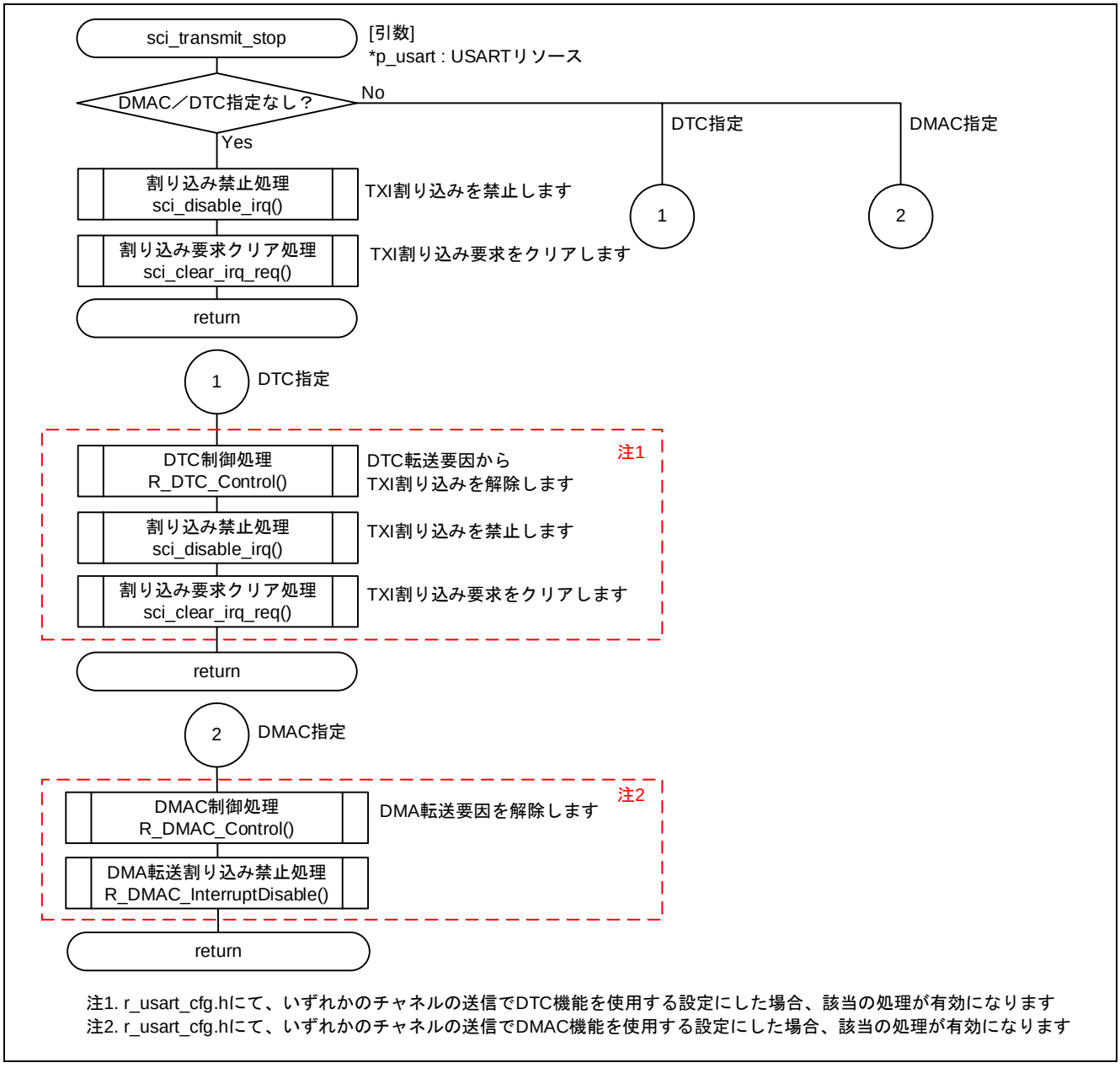


図 4-47 sci_transmit_stop 関数処理フロー

4.1.29 sci_receive_stop 関数

表 4-31 sci_receive_stop 関数仕様

書式	static void sci_receive_stop (st_usart_resources_t const * const p_usart)
仕様説明	受信を中断します
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	ー

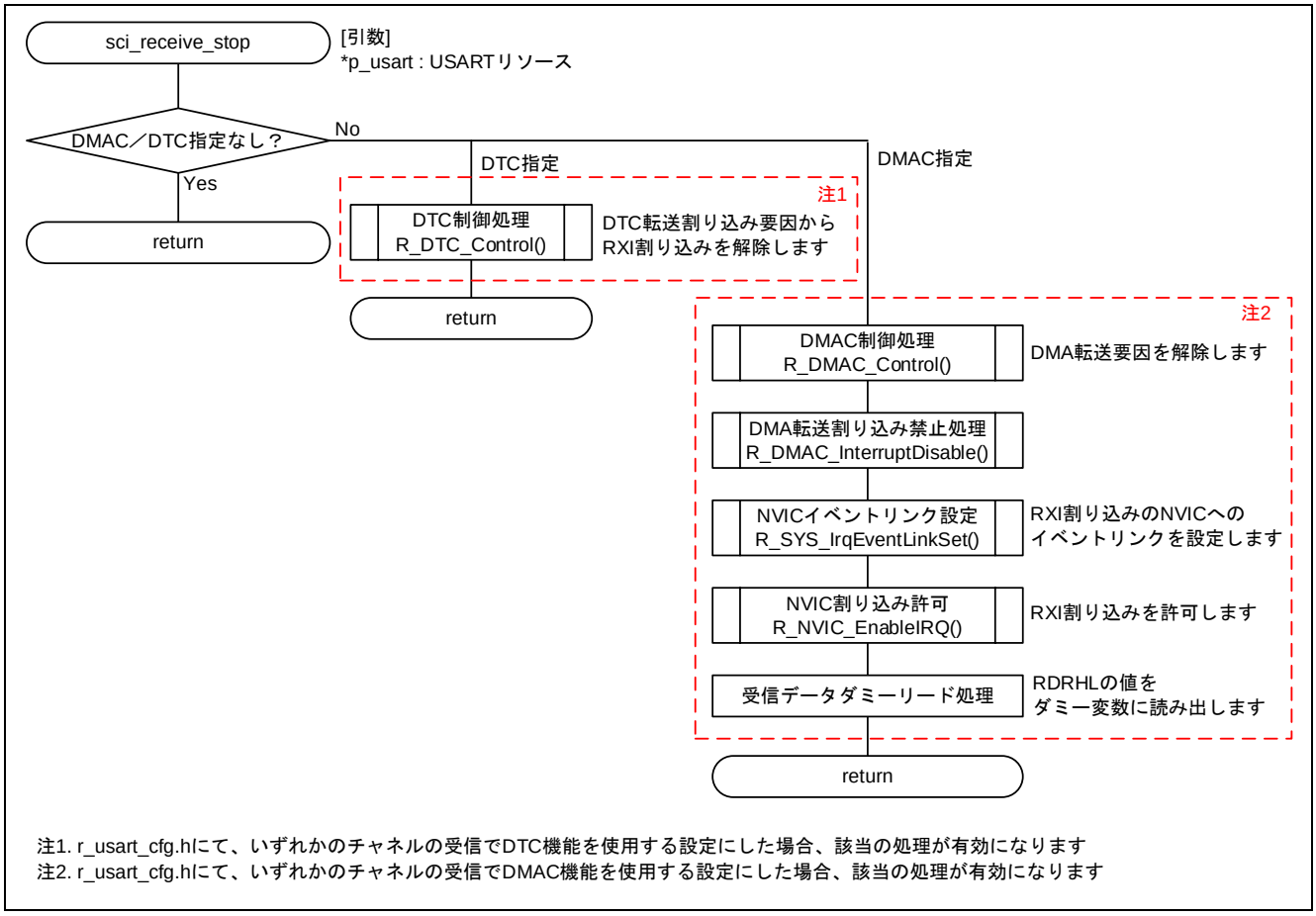


図 4-48 sci_receive_stop 関数処理フロー

4.1.30 dma_config_init 関数

表 4-32 dma_config_init 関数仕様

書式	static void dma_config_init(st_dma_transfer_data_cfg_t *p_cfg)
仕様説明	DMA ドライバ設定用構造体の 0 初期化
引数	st_dma_transfer_data_cfg_t *p_cfg: DMA ドライバ設定用構造体
戻り値	なし
備考	―

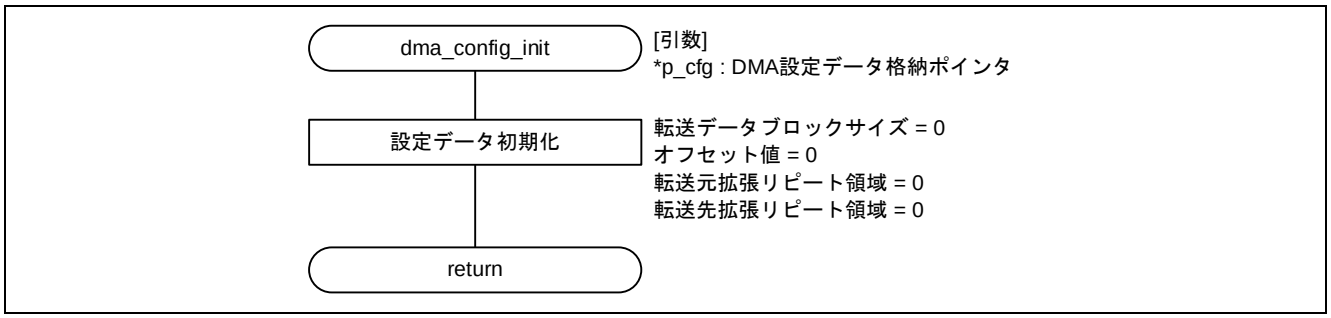


図 4-49 dma_config_init 関数処理フロー

4.1.31 txi_handler 関数

表 4-33 txi_handler 関数仕様

書式	static void txi_handler(st_usart_resources_t * const p_usart)
仕様説明	TXI 割り込み処理（送信処理に割り込み使用時）
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	送信処理に割り込みを使用した場合の TXI 割り込み処理です

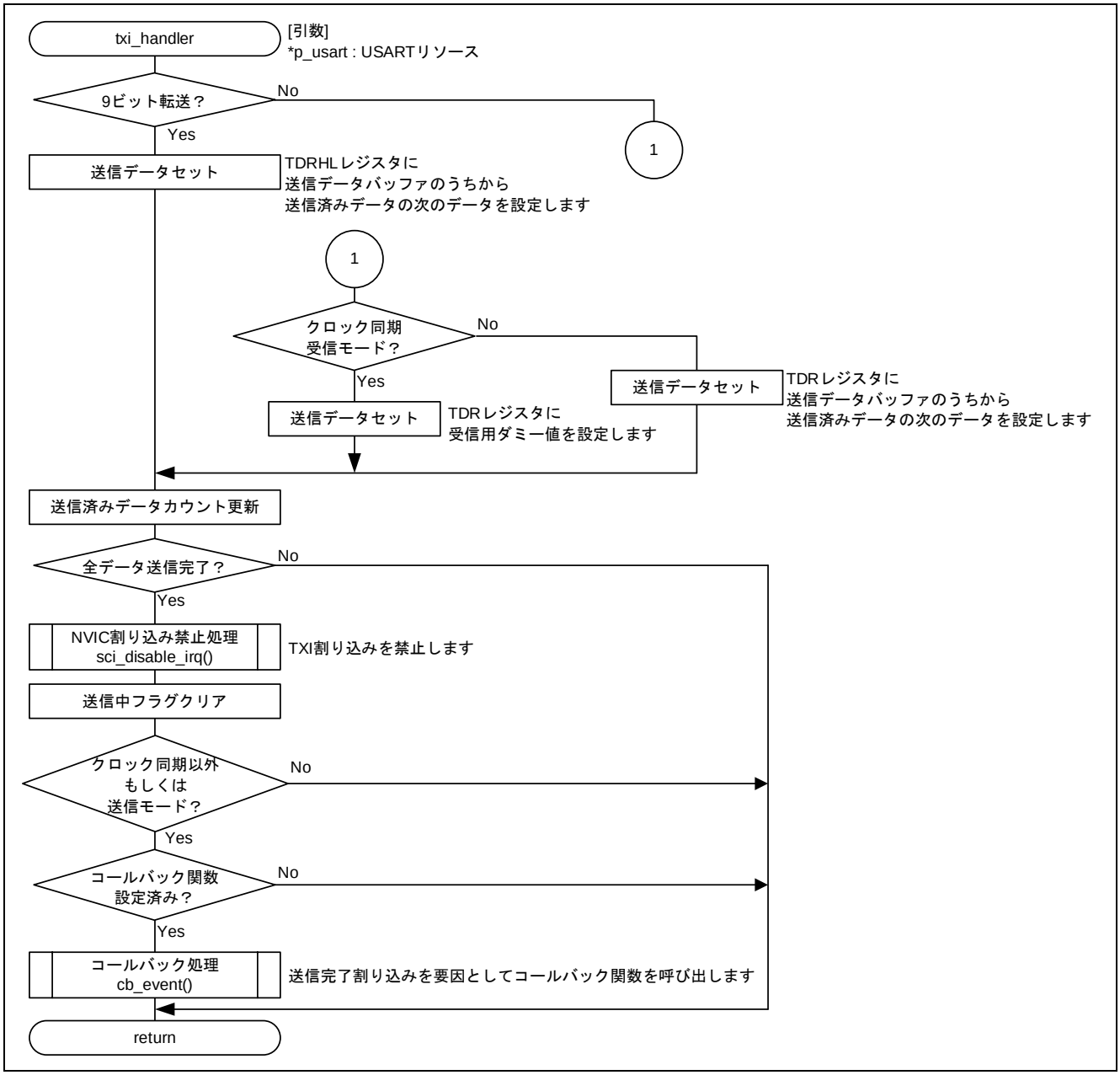


図 4-50 txi_handler 関数処理フロー

4.1.32 txi_dtc_handler 関数

表 4-34 txi_dtc_handler 関数仕様

書式	static void txi_dtc_handler(st_usart_resources_t * const p_usart)
仕様説明	TXI 割り込み処理（送信処理に DTC 使用時）
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	送信処理に DTC を使用した場合の TXI 割り込み処理です

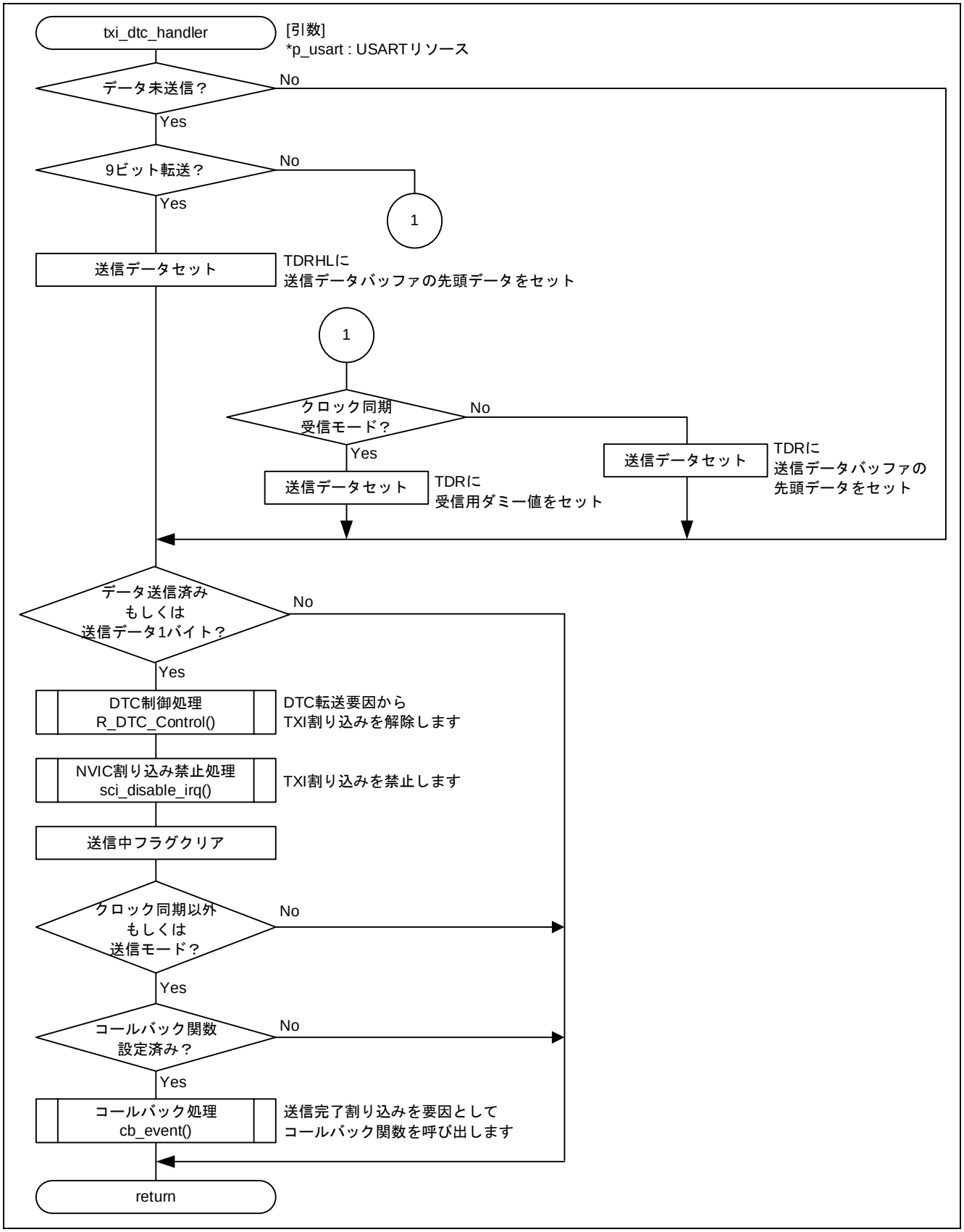


図 4-51 txi_dtc_handler 関数処理フロー

4.1.33 txi_dmac_handler 関数

表 4-35 txi_dmac_handler 関数仕様

書式	static void txi_dmac_handler(st_usart_resources_t * const p_usart)
仕様説明	TXI 割り込み処理（送信処理に DMAC 使用時）
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	送信処理に DMAC を使用した場合の TXI 割り込み処理です

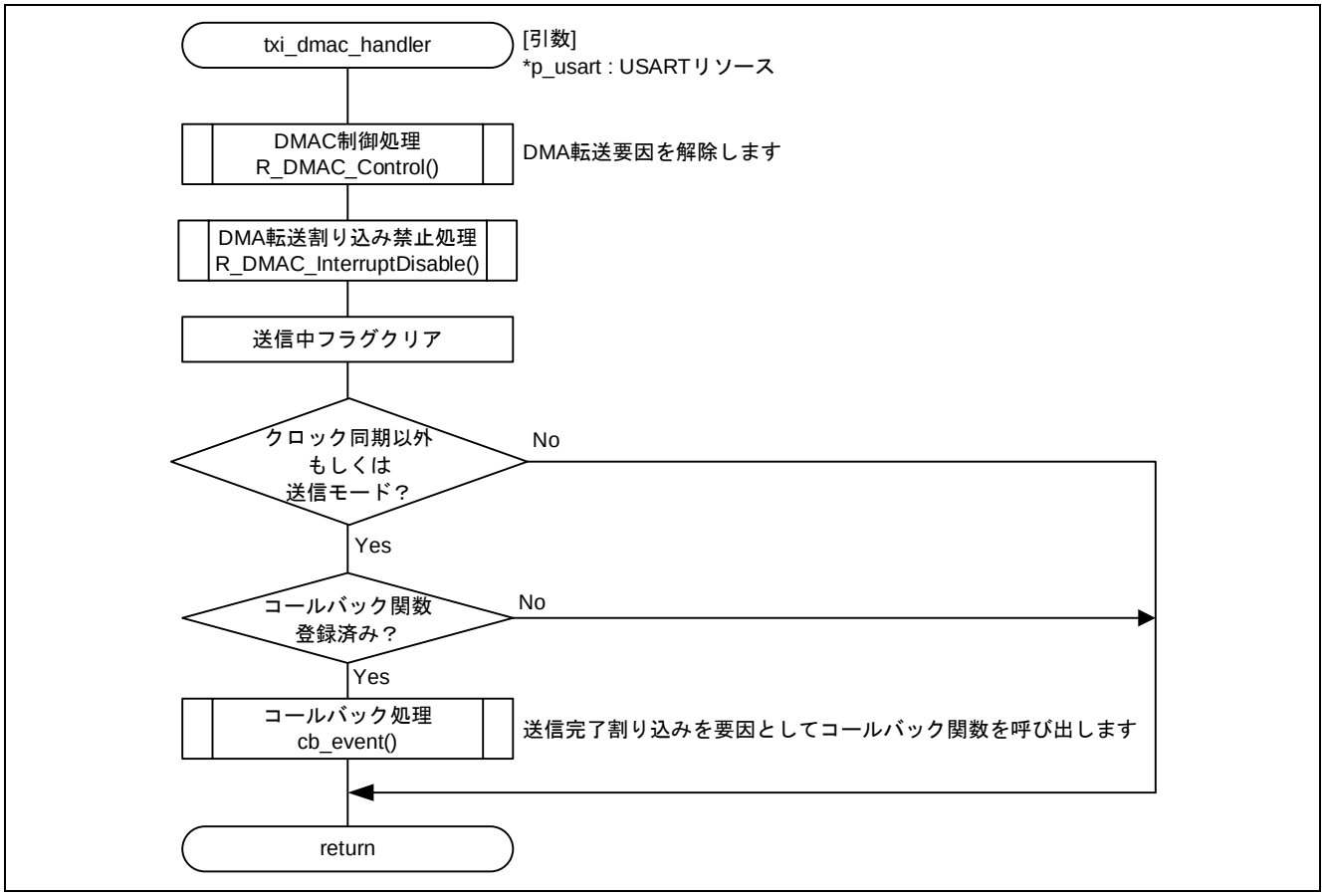


図 4-52 txi_dmac_handler 関数処理フロー

4.1.34 rxi_handler 関数

表 4-36 rxi_handler 関数仕様

書式	static void rxi_handler(st_usart_resources_t * const p_usart)
仕様説明	RXI 割り込み処理（受信処理に割り込み使用時）
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	受信処理に割り込みを使用した場合の RXI 割り込み処理です

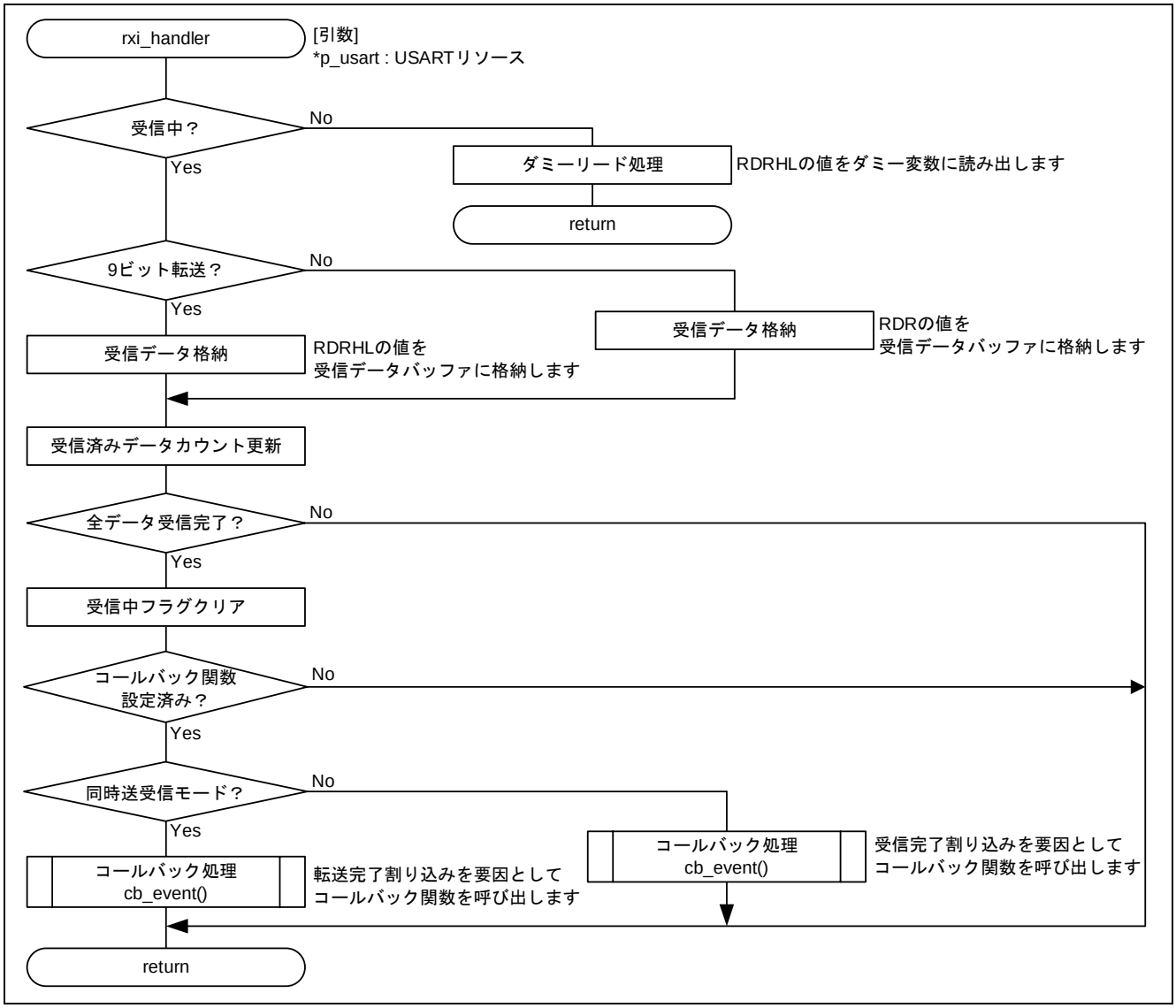


図 4-53 rxi_handler 関数処理フロー

4.1.35 rxi_dmac_handler 関数

表 4-37 rxi_dmac_handler 関数仕様

書式	static void rxi_dmac_handler(st_usart_resources_t * const p_usart)
仕様説明	RXI 割り込み処理（受信処理に DTC/DMAC 使用時）
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	受信処理に DTC または DMAC を使用した場合の RXI 割り込み処理です

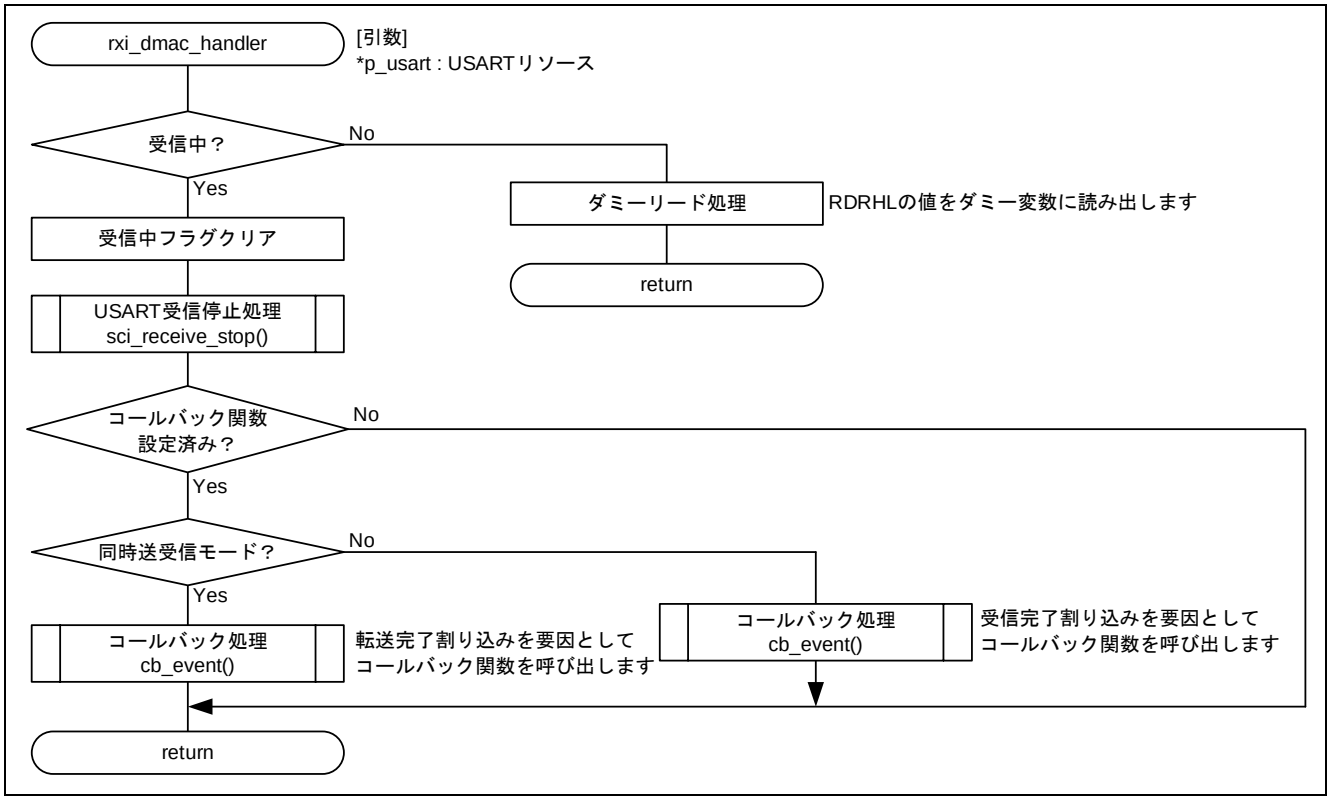


図 4-54 rxi_dmac_handler 関数処理フロー

4.1.36 eri_handler 関数

表 4-38 eri_handler 関数仕様

書式	static void eri_handler(st_usart_resources_t * const p_usart) // @suppress(“Function length”)
仕様説明	ERI 割り込み処理
引数	st_usart_resources_t * const p_usart : USART のリソース 制御対象の USART のリソースを指定します。
戻り値	なし
備考	―

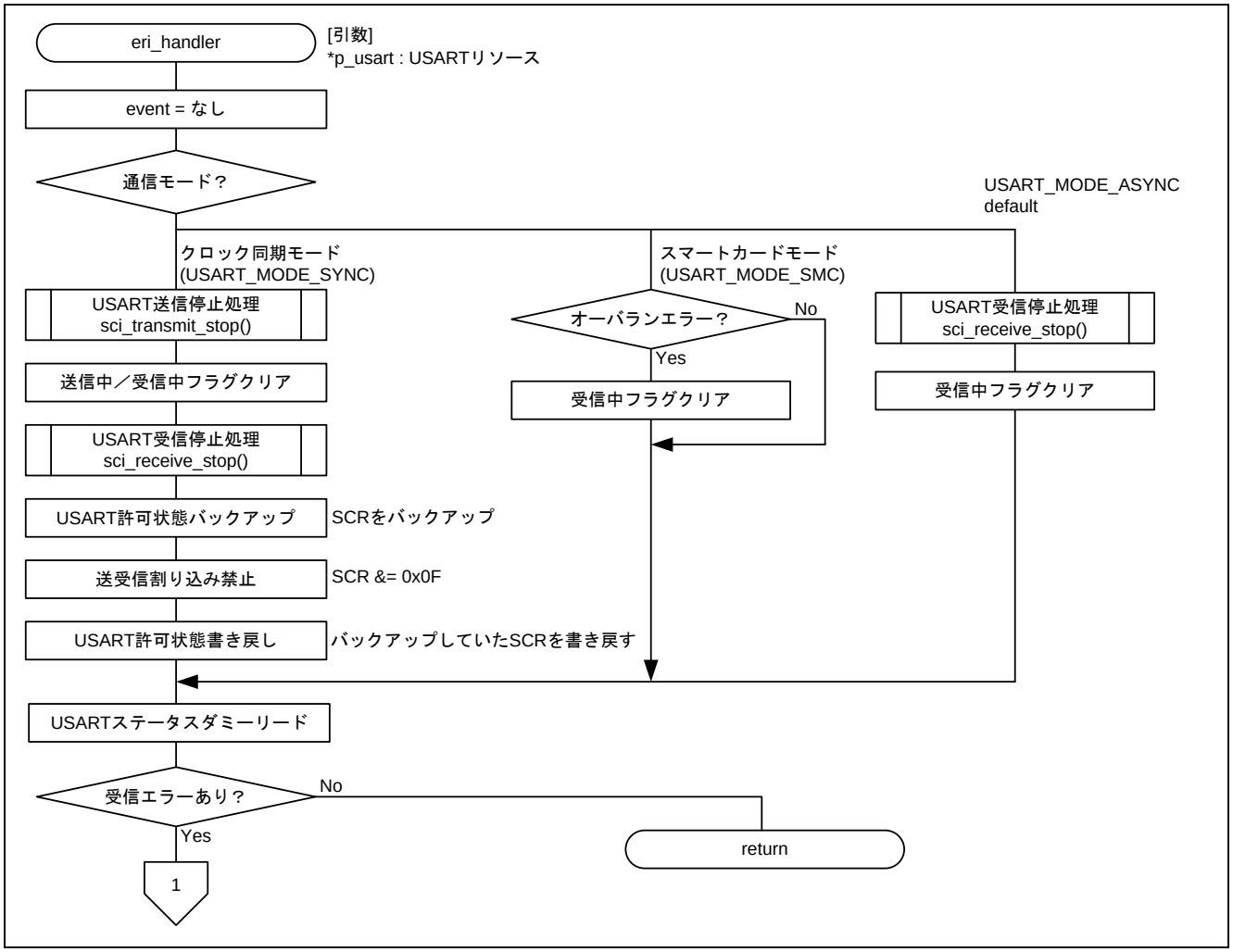


図 4-55 eri_handler 関数処理フロー(1/2)

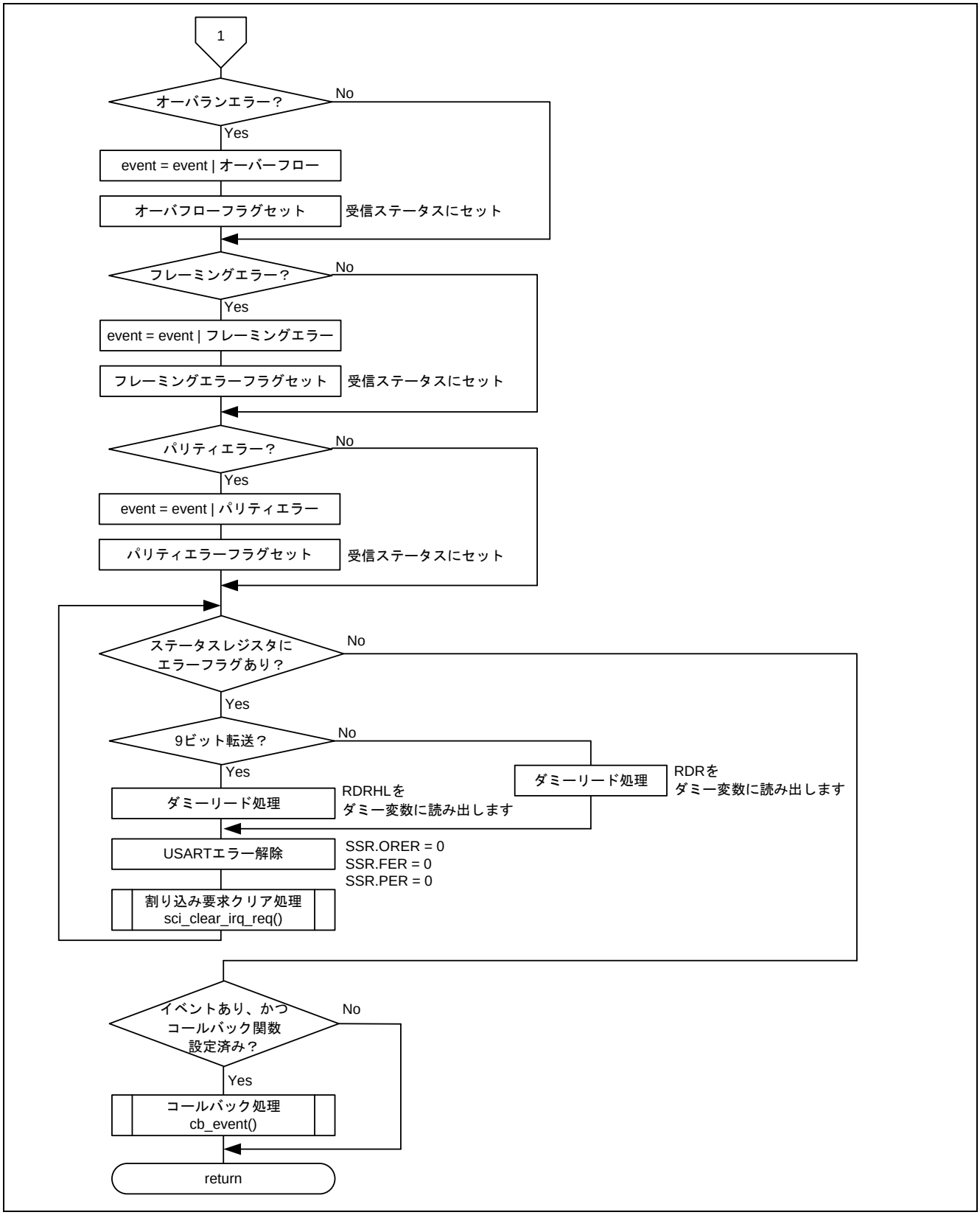


図 4-56 eri_handler 関数処理フロー(2/2)

4.2 マクロ／型定義

ドライバ内部で使用するマクロ／型定義を示します。

4.2.1 マクロ定義一覧

表 4-39 マクロ定義一覧

定義	値	内容
R_SCI0_ENABLE	(1)	SCI0 リソース有効定義
R_SCI1_ENABLE	(1)	SCI1 リソース有効定義
R_SCI2_ENABLE	(1)	SCI2 リソース有効定義
R_SCI3_ENABLE	(1)	SCI3 リソース有効定義
R_SCI4_ENABLE	(1)	SCI4 リソース有効定義
R_SCI5_ENABLE	(1)	SCI5 リソース有効定義
R_SCI9_ENABLE	(1)	SCI9 リソース有効定義
USART_SIZE_7	(0)	7 ビットデータ定義
USART_SIZE_8	(1)	8 ビットデータ定義
USART_SIZE_9	(2)	9 ビットデータ定義
USART_SSR_ORER_MASK	(0x20U)	ORER ビットマスク定義
USART_SSR_FER_MASK	(0x10U)	FER ビットマスク定義
USART_SSR_PER_MASK	(0x08U)	PER ビットマスク定義
USART_RCVR_ERR_MASK	(USART_SSR_ORER_MASK | USART_SSR_FER_MASK | USART_SSR_PER_MASK)	受信エラーマスク定義
USART_SSR_CLR_MASK	(0xC0U)	受信エラークリアマスク定義
USART_FLAG_INITIALIZED	(1U << 0)	USART 初期化済みフラグ定義
USART_FLAG_TX_AVAILABLE	(1U << 1)	送信可能フラグ定義
USART_FLAG_RX_AVAILABLE	(1U << 2)	受信可能フラグ定義
USART_FLAG_POWERED	(1U << 3)	モジュール解除済みフラグ定義
USART_FLAG_CONFIGURED	(1U << 4)	モード設定済みフラグ定義
USART_FLAG_TX_ENABLED	(1U << 5)	送信許可状態フラグ定義
USART_FLAG_RX_ENABLED	(1U << 6)	受信許可状態フラグ定義
USART_TXI0_IESR_VAL	(0x00000010)	TXI0 用 IELS ビット設定値
USART_RXI0_IESR_VAL	(0x00000010)	RXI0 用 IELS ビット設定値
USART_ERI0_IESR_VAL	(0x00000010)	ERI0 用 IELS ビット設定値
USART_TXI1_IESR_VAL	(0x0000001A)	TXI1 用 IELS ビット設定値
USART_RXI1_IESR_VAL	(0x0000001B)	RXI1 用 IELS ビット設定値
USART_ERI1_IESR_VAL	(0x0000001A)	ERI1 用 IELS ビット設定値
USART_TXI2_IESR_VAL	(0x0000001A)	TXI2 用 IELS ビット設定値
USART_RXI2_IESR_VAL	(0x0000001A)	RXI2 用 IELS ビット設定値
USART_ERI2_IESR_VAL	(0x00000019)	ERI2 用 IELS ビット設定値
USART_TXI3_IESR_VAL	(0x0000001C)	TXI3 用 IELS ビット設定値
USART_RXI3_IESR_VAL	(0x0000001C)	RXI3 用 IELS ビット設定値
USART_ERI3_IESR_VAL	(0x0000001B)	ERI3 用 IELS ビット設定値
USART_TXI4_IESR_VAL	(0x0000001B)	TXI4 用 IELS ビット設定値
USART_RXI4_IESR_VAL	(0x0000001B)	RXI4 用 IELS ビット設定値
USART_ERI4_IESR_VAL	(0x0000001A)	ERI4 用 IELS ビット設定値

表 4-40 マクロ定義一覧

定義	値	内容
USART_TXI5_IESR_VAL	(0x0000001D)	TXI5 用 IELS ビット設定値
USART_RXI5_IESR_VAL	(0x0000001D)	RXI5 用 IELS ビット設定値
USART_ERI5_IESR_VAL	(0x0000001C)	ERI5 用 IELS ビット設定値
USART_TXI9_IESR_VAL	(0x0000001C)	TXI9 用 IELS ビット設定値
USART_RXI9_IESR_VAL	(0x0000001C)	RXI9 用 IELS ビット設定値
USART_ERI9_IESR_VAL	(0x0000001C)	ERI9 用 IELS ビット設定値
USART_TXI0_DMAC_SOURCE_ID	(0x76)	TXI0 用 DELS ビット設定値
USART_RXI0_DMAC_SOURCE_ID	(0x75)	RXI0 用 DELS ビット設定値
USART_TXI1_DMAC_SOURCE_ID	(0x7C)	TXI1 用 DELS ビット設定値
USART_RXI1_DMAC_SOURCE_ID	(0x7B)	RXI1 用 DELS ビット設定値
USART_TXI2_DMAC_SOURCE_ID	(0x81)	TXI2 用 DELS ビット設定値
USART_RXI2_DMAC_SOURCE_ID	(0x80)	RXI2 用 DELS ビット設定値
USART_TXI3_DMAC_SOURCE_ID	(0x86)	TXI3 用 DELS ビット設定値
USART_RXI3_DMAC_SOURCE_ID	(0x85)	RXI3 用 DELS ビット設定値
USART_TXI4_DMAC_SOURCE_ID	(0x8B)	TXI4 用 DELS ビット設定値
USART_RXI4_DMAC_SOURCE_ID	(0x8A)	RXI4 用 DELS ビット設定値
USART_TXI5_DMAC_SOURCE_ID	(0x90)	TXI5 用 DELS ビット設定値
USART_RXI5_DMAC_SOURCE_ID	(0x8F)	RXI5 用 DELS ビット設定値
USART_TXI9_DMAC_SOURCE_ID	(0x95)	TXI9 用 DELS ビット設定値
USART_RXI9_DMAC_SOURCE_ID	(0x94)	RXI9 用 DELS ビット設定値
REG_PRV_VALUE_CKS0	(0)	CKS0 ビット位置定義
REG_PRV_VALUE_CKS1	(1)	CKS1 ビット位置定義
REG_PRV_VALUE_BGMD	(2)	BGMD ビット位置定義
REG_PRV_VALUE_ABCS	(3)	ABCS ビット位置定義
REG_PRV_VALUE_ABCSE	(4)	ABCSE ビット位置定義
REG_PRV_VALUE_BCP0	(5)	BCP0 ビット位置定義
REG_PRV_VALUE_BCP1	(6)	BCP1 ビット位置定義
REG_PRV_VALUE_BCP2	(7)	BCP2 ビット位置定義
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	DMAC/DTC ドライバ使用判定定義
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	DMAC/DTC 送信処理判定定義

表 4-41 マクロ定義一覧

定義	値	内容
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	DMAC/DTC 受信処理判定定義
USART_PRV_USED_DMAC_DRV	(USART_PRV_USED_DMAC_DTC_DRV & 0x00FF)	DMAC ドライバ使用判定定義
USART_PRV_USED_TX_DMAC_DRV	(USART_PRV_USED_TX_DMAC_DTC_DRV & 0x00FF)	DMAC 送信処理判定定義
USART_PRV_USED_RX_DMAC_DRV	(USART_PRV_USED_RX_DMAC_DTC_DRV & 0x00FF)	DMAC 受信処理判定定義
USART_PRV_USED_DTC_DRV	(USART_PRV_USED_DMAC_DTC_DRV & SCI_USED_DTC)	DTC ドライバ使用判定定義
USART_PRV_USED_TX_DTC_DRV	(USART_PRV_USED_TX_DMAC_DTC_DRV & SCI_USED_DTC)	DTC 送信処理判定定義
USART_PRV_USED_RX_DTC_DRV	(USART_PRV_USED_RX_DMAC_DTC_DRV & SCI_USED_DTC)	DTC 受信処理判定定義

4.2.2 e_usart_flow_t 定義

フロー制御状態を示す定義です。

表 4-42 e_usart_flow_t 定義一覧

定義	値	内容
USART_FLOW_CTS_DISABLE	0	フロー制御未使用
USART_FLOW_CTS_ENABLE	1	CTS 制御使用
USART_FLOW_RTS_ENABLE	2	RTS 制御使用

4.2.3 e_usart_mode_t 定義

動作モードを示す定義です。

表 4-43 e_usart_mode_t 定義一覧

定義	値	内容
USART_MODE_ASYNC	0	調歩同期モード
USART_MODE_SYNC	1	クロック同期モード
USART_MODE_SMC	2	スマートカードモード

4.2.4 e_usart_sync_t 定義

クロック同期モードにて送受信状態を示す定義です。

表 4-44 e_usart_sync_t 定義一覧

定義	値	内容
USART_SYNC_TX_MODE	0	送信モードで動作
USART_SYNC_RX_MODE	1	受信モードで動作
USART_SYNC_TX_RX_MODE	2	送受信モードで動作

4.2.5 e_usart_base_clk_t 定義

対象チャネルのベースクロックを示す定義です。

表 4-45 e_usart_base_clk_t 定義一覧

定義	値	内容
USART_BASE_PCLKA	0	ベースクロック：PCLKA
USART_BASE_PCLKB	1	ベースクロック：PCLKB

4.3 構造体定義

4.3.1 st_usart_resources_t 構造体

USART のリソースを構成する構造体です。

表 4-46 st_usart_resources_t 構造体

要素名	型	内容
*reg	volatile SCI2_Type	対象の SCI レジスタを示します
pin_set	r_pinset_t	端子設定用関数ポインタ
pin_clr	r_pinclr_t	端子解除用関数ポインタ
*info	st_usart_info_t	USART 状態情報
*cts_port	uint16_t	ソフトウェア制御による CTS 端子(ポートレジスタ)
cts_pin_no	uint8_t	ソフトウェア制御による CTS 端子(端子番号)
*rts_port	uint16_t	ソフトウェア制御による RTS 端子(ポートレジスタ)
rts_pin_no	uint8_t	ソフトウェア制御による RTS 端子(端子番号)
pclk	e_usart_base_clk_t	対象 SCI チャネルのベースクロック USART_BASE_PCLKA : PCLKA USART_BASE_PCLKB : PCLKB
lock_id	e_system_mcu_lock_t	SCI ロック ID
mstp_id	e_lpm_mstp_t	SCI モジュールストップ ID
txi_irq	IRQn_Type	TXI 割り込みの NVIC 割り当て番号
rxi_irq	IRQn_Type	RXI 割り込みの NVIC 割り当て番号
eri_irq	IRQn_Type	ERI 割り込みの NVIC 割り当て番号
txi_iesr_val	uint32_t	TXI 割り込みの IESR レジスタ設定値
rxi_iesr_val	uint32_t	RXI 割り込みの IESR レジスタ設定値
eri_iesr_val	uint32_t	ERI 割り込みの IESR レジスタ設定値
txi_priority	uint32_t	TXI 割り込み優先レベル
rxi_priority	uint32_t	RXI 割り込み優先レベル
eri_priority	uint32_t	ERI 割り込み優先レベル
*tx_dma_drv	DRIVER_DMA	送信用 DMA ドライバ 送信処理に割り込みを使用する場合は NULL が設定されます
tx_dma_source	uint16_t	TXI 用 DELS ビット設定値
*tx_dtc_info	st_dma_transfer_data_t	送信用 DTC 転送情報格納番地
*rx_dma_drv	DRIVER_DMA	受信用 DMA ドライバ 受信処理に割り込みを使用する場合は NULL が設定されます
rx_dma_source	uint16_t	RXI 用 DELS ビット設定値
*rx_dtc_info	st_dma_transfer_data_t	受信用 DTC 転送情報格納番地
txi_callback	system_int_cb_t	TXI コールバック関数
rxi_callback	system_int_cb_t	RXI コールバック関数
eri_callback	system_int_cb_t	ERI コールバック関数

4.3.2 st_usart_rx_status_t 構造体

USART の受信状態を管理するための構造体です。

表 4-47 st_usart_rx_status_t 構造体

要素名	型	内容
busy	uint8_t	受信中フラグ(0: 未受信、1: 受信中)
overflow	uint8_t	オーバフローエラーフラグ (0: オーバフローエラー未検出、1: オーバフローエラー検出)
framing_error	uint8_t	フレーミングエラーフラグ (0: フレーミングエラー未検出、1: フレーミングエラー検出)
parity_error	uint8_t	パリティエラーフラグ (0: パリティエラー未検出、1: パリティエラー検出)

4.3.3 st_usart_transfer_info_t 構造体

USART の送受信情報を管理するための構造体です。

表 4-48 st_usart_transfer_info_t 構造体

要素名	型	内容
rx_num	uint32_t	受信サイズ
tx_num	uint32_t	送信サイズ
*rx_buf	void	受信バッファ
*tx_buf	void	送信バッファ
rx_cnt	uint32_t	受信カウント
tx_cnt	uint32_t	送信カウント
tx_def_val	uint16_t	クロック同期受信モードでのダミー送信データ
rx_dump_val	uint8_t	ダミーリード用バッファ
send_active	uint8_t	送信中フラグ(0: 未送信、1: 送信中)
sync_mode	e_usart_sync_t	クロック同期動作モード(クロック同期モードでのみ有効) USART_SYNC_TX_MODE : 送信モードで動作 USART_SYNC_RX_MODE : 受信モードで動作 USART_SYNC_TX_RX_MODE : 送受信モードで動作

4.3.4 st_usart_info_t 構造体

USART の情報を管理するための構造体です。

表 4-49 st_usart_info_t 構造体

要素名	型	内容
cb_event	ARM_USART_SignalEvent_t	イベント発生時のコールバック関数 NULL の場合はコールバック関数実行しない
rx_status	st_usart_rx_status_t	USART 受信状態
tx_status	st_usart_transfer_info_t	USART 送受信情報
mode	e_usart_mode_t	動作モード USART_MODE_ASYNC : 調歩同期モード USART_MODE_SYNC : クロック同期モード USART_MODE_SMC : スマートカードモード
data_size	uint8_t	送受信データサイズ USART_SIZE_7 : 7 ビット長 USART_SIZE_8 : 8 ビット長 USART_SIZE_9 : 9 ビット長
flow_mode	e_usart_flow_t	フロー制御 USART_FLOW_CTS_DISABLE : フロー制御未使用 USART_FLOW_CTS_ENABLE : CTS 制御使用 USART_FLOW_RTS_ENABLE : RTS 制御使用
flags	uint16_t	ドライバ状態フラグ b0 : ドライバ初期化状態(0:未初期化、1:初期化済み) b1 : 送信可否状態(0:送信不可、1:送信可) b2 : 受信可否状態(0:受信不可、1:受信可) b3 : モジュールストップ状態 (0:モジュールストップ状態、1:モジュールストップ解除) b4 : USART モード設定済み状態(0:未設定、1:設定済み) b5 : 送信許可状態(0:送信禁止状態、1:送信許可状態) b6 : 受信許可状態(0:受信禁止状態、1:受信許可状態)
baudrate	uint32_t	ボーレート設定

4.3.5 st_sci_reg_set_t 構造体

レジスタ設定用バッファの構造体です。

表 4-50 st_sci_reg_set_t 構造体

要素名	型	内容
smr	uint8_t	SMR レジスタ設定バッファ
scr	uint8_t	SCR レジスタ設定バッファ
brr	uint8_t	BRR レジスタ設定バッファ
scmr	uint8_t	SCMR レジスタ設定バッファ
semr	uint8_t	SEMR レジスタ設定バッファ
spmr	uint8_t	SPMR レジスタ設定バッファ
mddr	uint8_t	MDDR レジスタ設定バッファ
flow_mode	e_usart_flow_t	フロー制御設定バッファ
data_size	uint8_t	データサイズ設定バッファ

4.3.6 st_baud_divisor_t 構造体

ボーレート算出用テーブルの構造際です。

表 4-51 st_baud_divisor_t 構造体

要素名	型	内容
divisor	int64_t	分周比
reg_value	uint16_t	レジスタ設定値

4.4 データテーブル定義

USART ドライバの処理で使用する主なデータテーブル定義を示します。

4.4.1 ボーレート算出用データテーブル

ボーレート算出用データテーブルは st_baud_divisor_t 構造体で定義され、divisor 要素に分周比、reg_value 要素にレジスタの設定値が格納されます。

reg_value 要素にはボーレートに関わるビットの設定値を格納します。reg_value 要素の構成を図 4-57 に示します。

図 4-57 reg_value 要素の構成

ボーレート算出用テーブルは各モードによって異なります。

調歩同期モードでのボーレート算出テーブルを表 4-52 に、クロック同期モードでのボーレート算出テーブルを表 4-53 に、スマートカードモードでのボーレート算出テーブルを表 4-54 に示します。

表 4-52 調歩同期モードでのボーレート算出テーブル(gs_async_baud)

分周比	レジスタ設定値						内容
	CKS[1:0]	BGMD	ABCS	ABCSE	BCP[1:0]	BCP2	
6	0	0	0	1	0	0	6 分周時のレジスタ設定値
8	0	1	1	0	0	0	8 分周時のレジスタ設定値
16	0	1	0	0	0	0	16 分周時のレジスタ設定値
24	1	0	0	1	0	0	24 分周時のレジスタ設定値
32	1	1	1	0	0	0	32 分周時のレジスタ設定値
64	1	1	0	0	0	0	64 分周時のレジスタ設定値
96	2	0	0	1	0	0	96 分周時のレジスタ設定値
128	2	1	1	0	0	0	128 分周時のレジスタ設定値
256	2	1	0	0	0	0	256 分周時のレジスタ設定値
384	3	0	0	1	0	0	384 分周時のレジスタ設定値
512	3	1	1	0	0	0	512 分周時のレジスタ設定値
1024	3	1	0	0	0	0	1024 分周時のレジスタ設定値
2048	3	0	0	0	0	0	2048 分周時のレジスタ設定値

表 4-53 クロック同期モードでのボーレート算出テーブル(gs_sync_baud)

分周比	レジスタ設定値						内容
	CKS[1:0]	BGMD	ABCS	ABCSE	BCP[1:0]	BCP2	
4	0	0	0	0	0	0	4 分周時のレジスタ設定値
16	1	0	0	0	0	0	16 分周時のレジスタ設定値
64	2	0	0	0	0	0	64 分周時のレジスタ設定値
256	3	0	0	0	0	0	256 分周時のレジスタ設定値

表 4-54 スマートカードモードでのボーレート算出テーブル(gs_smc_baud)

分周比	レジスタ設定値						内容
	CKS[1:0]	BGMD	ABCS	ABCSE	BCP[1:0]	BCP2	
64	0	0	0	0	0	1	64 分周時のレジスタ設定値
128	0	0	0	0	1	1	128 分周時のレジスタ設定値
186	0	0	0	0	0	0	186 分周時のレジスタ設定値
256	1	0	0	0	0	1	256 分周時のレジスタ設定値
372	0	0	0	0	2	0	372 分周時のレジスタ設定値
512	1	0	0	0	1	1	512 分周時のレジスタ設定値
744	1	0	0	0	0	0	744 分周時のレジスタ設定値
1024	2	0	0	0	0	1	1024 分周時のレジスタ設定値
1488	1	0	0	0	2	0	1488 分周時のレジスタ設定値
2048	2	0	0	0	1	1	2048 分周時のレジスタ設定値
2976	2	0	0	0	0	0	2976 分周時のレジスタ設定値
4096	3	0	0	0	0	1	4096 分周時のレジスタ設定値
5952	2	0	0	0	2	0	5952 分周時のレジスタ設定値
8192	3	0	0	0	1	1	8192 分周時のレジスタ設定値
11904	3	0	0	0	0	0	11904 分周時のレジスタ設定値
16384	3	0	0	0	1	0	16384 分周時のレジスタ設定値
23808	3	0	0	0	2	0	23808 分周時のレジスタ設定値
32768	3	0	0	0	3	1	32768 分周時のレジスタ設定値
47616	3	0	0	0	2	1	47616 分周時のレジスタ設定値
65536	3	0	0	0	3	0	65536 分周時のレジスタ設定値

4.5 外部関数の呼び出し

USART ドライバ API から呼び出される外部関数を示します。

表 4-55 USART ドライバ API から呼び出す外部関数と呼び出し条件(1/3)

API	呼び出し関数	条件（注）
Initialize	R_SYS_ResourceLock	なし
	R_NVIC_GetPriority	なし
	R_NVIC_SetPriority	なし
	R_SYS_IrqEventLinkSet	なし
	R_NVIC_ClearPendingIRQ	なし
	R_SYS_IrqStatusClear	なし
	R_DMAC_Open	送信処理または受信処理に DMAC ドライバを使用した場合
	R_DMAC_InterruptEnable	
	R_DMAC_Close	送信処理または受信処理に DMAC ドライバを使用した場合かつ初期化処理に失敗した場合
	R_DTC_Open	送信処理または受信処理に DTC ドライバを使用した場合
	R_DTC_Close	送信処理または受信処理に DTC ドライバを使用した場合かつ初期化処理に失敗した場合
Uninitialize	R_LPM_ModuleStart	モジュールストップ状態で Uninitialize 関数実行時
	R_LPM_ModuleStop	なし
	R_SYS_ResourceUnlock	なし
	R_NVIC_ClearPendingIRQ	なし
	R_SYS_IrqStatusClear	なし
	R_NVIC_DisableIRQ	なし
	R_SCI_Pinclr_CHn(n=0～5,9)	なし
	R_DMAC_Close	送信処理または受信処理に DMAC ドライバを使用した場合
	R_DTC_Close	送信処理または受信処理に DTC ドライバを使用した場合
PowerControl	R_LPM_ModuleStart	ARM_POWER_FULL 指定時（モジュールストップ解除）
	R_LPM_ModuleStop	ARM_POWER_OFF 指定時（モジュールストップ遷移）
	R_NVIC_ClearPendingIRQ	
	R_SYS_IrqStatusClear	
	R_NVIC_DisableIRQ	
Send	R_NVIC_EnableIRQ	なし
	R_DMAC_Create	送信処理に DMAC ドライバを使用した場合
	R_DMAC_InterruptEnable	
	R_DMAC_Control	
	R_DTC_Create	送信処理に DTC ドライバを使用した場合
	R_DTC_Control	

注　条件なしの場合でも、パラメータチェックによるエラー終了発生時には呼び出し関数が実行されない可能性があります。

表 4-56 USART ドライバ API から呼び出す外部関数と呼び出し条件(2/3)

API	呼び出し関数	条件（注）
Receive	R_NVIC_EnableIRQ	なし
	R_DMAC_Create	受信処理に DMAC ドライバを使用した場合、 またはクロック同期モード（送信許可状態）で送信処理に DMAC ドライバを使用した場合
	R_DMAC_InterruptEnable	
	R_DMAC_Control	
	R_DMAC_InterruptDisable	クロック同期モード（送信許可状態）で送信処理に DMAC ドライバを使用し、かつ送信処理で DMAC 設定に失敗した場合
	R_SYS_IrqEventLinkSet	受信処理または送信処理に DMAC ドライバを使用した場合
	R_NVIC_ClearPendingIRQ	
	R_SYS_IrqStatusClear	
	R_NVIC_DisableIRQ	
	R_DTC_Create	受信処理に DTC ドライバを使用した場合、 またはクロック同期モード（送信許可状態）で送信処理に DTC ドライバを使用した場合
	R_DTC_Control	
Transfer	R_NVIC_EnableIRQ	なし
	R_DMAC_Create	送信処理または受信処理に DMAC ドライバを使用した場合
	R_DMAC_InterruptEnable	
	R_DMAC_Control	
	R_SYS_IrqEventLinkSet	
	R_NVIC_ClearPendingIRQ	
	R_SYS_IrqStatusClear	
	R_NVIC_DisableIRQ	
	R_DMAC_InterruptDisable	送信処理に DMAC ドライバを使用し、かつ送信処理で DMAC 設定に失敗した場合
	R_DTC_Create	受信処理に DTC ドライバを使用した場合、 またはクロック同期モード（送信許可状態）で送信処理に DTC ドライバを使用した場合
	R_DTC_Control	
GetTxCount	R_DMAC_GetTransferByte	送信処理に DMAC ドライバを使用し、かつ送信サイズが 1 バイトでない場合
	R_DTC_GetTransferByte	送信処理に DTC ドライバを使用した場合
GetRxCount	R_DMAC_GetTransferByte	受信処理に DMAC ドライバを使用した場合
	R_DTC_GetTransferByte	受信処理に DTC ドライバを使用した場合

注　条件なしの場合でも、パラメータチェックによるエラー終了発生時には呼び出し関数が実行されない可能性があります。

表 4-57 USART ドライバ API から呼び出す外部関数と呼び出し条件(3/3)

API	呼び出し関数	条件
Control	R_SYS_PeripheralClockFreqGet R_SYS_SystemClockFreqGet	以下のいずれかのコマンドを実行した場合 ・ARM_USART_MODE_ASYNCHRONOUS ・ARM_USART_MODE_SYNCHRONOUS_MASTER ・ARM_USART_MODE_SMART_CARD
	R_NVIC_ClearPendingIRQ R_SYS_IrqStatusClear	以下のいずれかのコマンドを実行した場合 ・ARM_USART_ABORT_SEND ・ARM_USART_ABORT_RECEIVE ・ARM_USART_ABORT_TRANSFER
	R_DMAC_Control R_DMAC_InterruptDisable	送信処理、または受信処理に DMAC ドライバを使用し、以下のいずれかのコマンドを実行した場合 ・ARM_USART_ABORT_SEND ・ARM_USART_ABORT_RECEIVE ・ARM_USART_ABORT_TRANSFER
	R_SYS_IrqEventLinkSet R_NVIC_EnableIRQ	受信処理に DMAC ドライバを使用し、以下のいずれかのコマンドを実行した場合 ・ARM_USART_ABORT_RECEIVE ・ARM_USART_ABORT_TRANSFER
	R_NVIC_ClearPendingIRQ R_SYS_IrqStatusClear	以下のいずれかのコマンドで受信を禁止にした場合 ・ARM_USART_ABORT_RECEIVE ・ARM_USART_ABORT_TRANSFER
	R_SCI_Pinset_CHn(n=0～5,9) R_NVIC_EnableIRQ	以下のいずれかのコマンドで送信、または受信を許可にした場合 ・ARM_USART_CONTROL_TX ・ARM_USART_CONTROL_RX ・ARM_USART_CONTROL_TX_RX
	R_SCI_Pinclr_CHn(n=0～5,9)	以下のいずれかのコマンドで送信および受信が共に禁止になった場合 ・ARM_USART_CONTROL_TX ・ARM_USART_CONTROL_RX ・ARM_USART_CONTROL_TX_RX
	R_NVIC_DisableIRQ	以下のいずれかのコマンドで送信または受信が禁止になった場合 ・ARM_USART_CONTROL_TX ・ARM_USART_CONTROL_RX ・ARM_USART_CONTROL_TX_RX
GetStatus	-	-
SetModemControl	-	-
GetModemStatus	-	-
GetVersion	-	-
GetCapabilities	-	-

5. 使用上の注意

5.1 引数について

各関数の引数に使用する構造体は、使用前にすべての要素を 0 で初期化してください。

5.2 NVIC への USART 割り込み登録

通信制御で使用する割り込みは、r_system_cfg.h にてネスト型ベクタ割り込みコントローラ（以下、NVIC）に登録する必要があります。

詳細は「2.4 通信制御および NVIC 割り込み設定」を参照してください。

5.3 電源オープン制御レジスタ(VOCR)設定について

本ドライバは、電源オープン制御レジスタ（VOCR）の設定を行った上で使用してください。

VOCR レジスタは、電源供給されていない電源ドメインから不定な入力が入ることを阻止するレジスタです。このため、VOCR レジスタはリセット後、入力信号を遮断する設定になっています。この状態では入力信号がデバイス内部に伝搬されません。詳細は「RE01 1500KB 、256KB グループ CMSIS パッケージを用いた開発スタートアップガイド（r01an4660）」の「IO 電源ドメイン不定値伝搬抑止制御」を参照してください。

5.4 クロック同期、スマートカード通信にて受信を使用する場合の設定について

クロック同期で送信・受信を同時に行う場合は、以下の手順で TE、RE の許可を行ってください。H/W の制限で TE、RE を同時に許可する必要があります。以下の手順に従わない場合、先に設定した側のみ設定されます。

受信のみを行う場合も、ダミーデータの書き込みを行うため同様の手順で行ってください。クロック同期にて受信を使用する場合の設定例を図 5-1 に示します。

```
#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;
    /* クロック同期 マスタモード */
    arg = ARM_USART_MODE_SYNCHRONOUS_MASTER |
          ARM_USART_CPOL0 | ARM_USART_CPHA0 |
          ARM_USART_FLOW_CONTROL_NONE,;

    (void)gsp_sci0_dev->Initialize(usart_callback);              /* USART ドライバ初期化 */
    (void)gsp_sci0_dev->PowerControl(ARM_POWER_FULL);    /* USART のモジュールストップ解除 */
    (void)gsp_sci0_dev->Control(arg, 100000);                /* クロック同期マスタモード
(100kbps) */
    (void)gsp_sci0_dev->Control(ARM_USART_CONTROL_TX_RX,1);  /* 送信および受信を同時に許可 */

    (void) gsp_sci0_dev->Receive(rx_data, 6);  /* 受信開始 */
    while(1);
}

/**********************************************************************************************
* callback function
**********************************************************************************************/
static void usart_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_USART_EVENT_SEND_COMPLETE:
        {
            /* 正常に送信完了した場合の処理を記述 */
        }
        break;

        case ARM_USART_EVENT_RECEIVE_COMPLETE:
        {
            /* 正常に受信完了した場合の処理を記述 */
        }
        break;

        case ARM_USART_EVENT_TRANSFER_COMPLETE:
        {
            /* 正常に送受信完了した場合の処理を記述 */
        }
        break;

        default:
        {
            /* 通信異常が発生した場合の処理を記述 */
        }
        break;
    }
}
```

図 5-1 クロック同期にて受信を使用する場合の設定例

5.5 端子設定について

本ドライバで使用する端子は、pin.c の R_SCI_Pinset_CHn(n=0～5,9)関数で設定、R_SCI_Pinclr_CHn 関数で解放されます。R_SCI_Pinset_CHn 関数は Control 関数で送信または受信が許可状態になったときに呼び出されます。R_SCI_Pinclr_CHn 関数は Control 関数、PowerControl 関数、または Uninitialize 関数で送受信が禁止状態になったときに呼び出されます。

使用する端子は、pin.c の R_SCI_Pinset_CHn、R_SCI_Pinclr_CHn (n=0～5,9)関数内を修正して選択してください。SCI0 を使用する場合の端子設定変更例を図 5-2～図 5-4 に示します。

```
/*****************************************************************************//**
* @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;

/* P703 を TXD0 用端子に設定 */
    /* 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;
```

図 5-2 pin.c での端子設定例(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;

/* P702 を RXD0 用端子に設定 */
    /* 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() */
```

図 5-3 pin.c での端子設定例(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;

/* RXD0 端子を解放 */
    /* 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;

/* TXD0 端子を解放 */
    /* 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() */
```

図 5-4 pin.c での端子設定例(3/3)

5.6 送信制御に DMAC 制御を使用した場合の注意事項

送信制御に DMAC を使用した場合、1 バイト目の送信データは Send 関数内で書き込みます。そのため、1 バイト送信のみ実施する場合は、Send 関数内でコールバック関数が呼び出されます。

コールバック関数で再度 1 バイト送信を行うと、再帰関数（コールバック関数内でコールバック関数を呼び出す）となります。

5.7 SetModemControl 関数による RTS 制御について

SetModemControl 関数を使用してソフトウェアによる RTS を制御する場合、r_usart_cfg.h ファイルの USARTn_RTS_PORT、USARTn_RTS_PIN(n=0～5、9)にて RTS として使用する端子を設定してください。また、Control 関数で動作モードを選択する際のフロー制御には、ARM_USART_FLOW_CONTROL_NONE,（CTS/RTS 未使用）または、ARM_USART_FLOW_CONTROL_CTS（CTS 機能を使用）にしてください。（H/W による RTS 制御に設定しないでください）

USART0 にてソフトウェアによる RTS 制御端子を PORT508 に設定する場合の例を図 5-5 に示します。

```
/* 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
```

図 5-5 r_usart_cfg.h でのソフトウェアによる RTS 制御端子設定例

5.8 GetModemStatus 関数による CTS 端子状態の取得について

GetModemStatus 関数を使用してソフトウェアによる CTS 端子状態を取得する場合、r_usart_cfg.h ファイルの USARTn_CTS_PORT、USARTn_CTS_PIN(n=0～5、9)にて CTS として使用する端子を設定してください。また、Control 関数で動作モードを選択する際のフロー制御には、ARM_USART_FLOW_CONTROL_NONE,（CTS/RTS 未使用）または、ARM_USART_FLOW_CONTROL_RTS（RTS 機能を使用）にしてください。（H/W による CTS 制御に設定しないでください）

USART0 にて GetModemStatus 関数で使用する CTS 端子を PORT509 に設定する場合の例を図 5-6 に示します。

```
/* 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
```

図 5-6 r_usart_cfg.h での GetModemStatus 関数で使用する CTS 端子の設定例

5.9 DTC 使用時の注意

本ドライバの r_usart_cfg.h ファイルにて送信制御もしくは受信制御に DTC を選択した場合、表 5-1 の示す API で条件が満たされると DTC ドライバの R_DTC_Close 関数が実行されます。R_DTC_Close 関数が実行されると、すべての DTC 転送要因が解放されます。

複数のドライバで DTC を使用している場合、R_DTC_Close 関数実行時にすべての DTC 設定が解除され、DTC 転送が停止する問題が発生します。DTC 使用中に R_DTC_Close 関数が実行されないよう注意してください。

表 5-1 に、本ドライバの API 処理内で、R_DTC_Close 関数を実行する条件を示します。

表 5-1 R_DTC_Close 関数を実行する関数一覧

関数	R_DTC_Close 関数実行条件
ARM_USART_Initialize	ARM_DRIVER_ERROR 発生時
ARM_USART_Uninitialize	ARM_USART_Uninitialize 関数実行時

6. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

RE01 1500KB グループ ユーザーズマニュアル　ハードウェア編 R01UH0796
RE01 256KB グループ ユーザーズマニュアル　ハードウェア編 R01UH0894

（最新版をルネサス エレクトロニクスホームページから入手してください。）

RE01 グループ CMSIS Package スタートアップガイド

RE01 1500KB、256KB グループ CMSIS パッケージを用いた開発スタートアップガイド R01AN4660

（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース

（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境

（最新版をルネサス エレクトロニクスホームページから入手してください。）

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Oct.10.19	ー	初版
1.01	Nov.28.2019	15, 164~166	pin.c のデフォルト端子設定コメントアウト化にともなう修正
1.03	Feb.18.2020	ー 128, 132,133 30,31, 38~40, 46~48, 51~53, 59~61, 64~67 プログラム	誤記修正 端子設定手順のフローをプログラムに合わせて修正 送信、送受信タイミング図の説明文誤記修正 以下 2 つの修正を実施 ・端子設定時に数サイクルの間、出力が HI-z になる問題を修正 - ドライバ内部の端子設定手順を見直し ・USART CH9 の受信制御に DTC を選択したとき、受信完了割り込みが発生しない問題を修正 - 内部関数 sci9_rxi_interrupt()内の判定式を修正
1.04	Mar.5.2020	ー プログラム 256KB	256KB グループに対応 256KB の IO デファインにあわせて修正 ・TDR レジスタへのアクセス方法を変更 ・RDR レジスタへのアクセス方法を変更 ・TDRHL レジスタへのアクセス方法を変更 ・RDRHL レジスタへのアクセス方法を変更
1.05	Apr.17.2020	プログラム	DMAC および DTC ドライバがない状態でビルドできる形に構成を変更
1.06	Nov.05.2020	169 ー	"5.9　DTC 使用時の注意"追加 誤記修正

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策

   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。

2. 電源投入時の処置

   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. 電源オフ時における入力信号

   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。

4. 未使用端子の処理

   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

5. クロックについて

   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

6. 入力端子の印加波形

   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。

7. リザーブアドレス（予約領域）のアクセス禁止

   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

8. 製品間の相違について

   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。