RE01 1500KB、256KB グループ

CMSIS ドライバ SPI 仕様書

要旨

本書では、RE01 1500KB、256KB グループ CMSIS software package の SPI ドライバ（以下、SPI 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

SPI 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_SPI ドライバで使用する周辺機能

周辺機能	内容
シリアルペリフェラルインタフェース(SPI)	SPI を使用し、SPI 通信（4 線式）またはクロック同期式通信（3 線式）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、SPI データレジスタ(SPDR)へのデータ書き込みおよび SPDR 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、SPI データレジスタ(SPDR)へのデータ書き込みおよび SPDR 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

SPI 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_SPI.h”と、ベンダ独自ファイル格納ディレクトリ内の ”r_spi_cmsis_api.c”、”r_spi_cmsis_api.h”、”r_spi_cfg.h” 、”pin.c”、”pin.h”の 6 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_SPI ドライバ 各ファイルの役割

ファイル名	内容
Driver_SPI.h	CMSIS Driver 標準ヘッダファイルです
r_spi_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 SPI 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_spi_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_spi_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

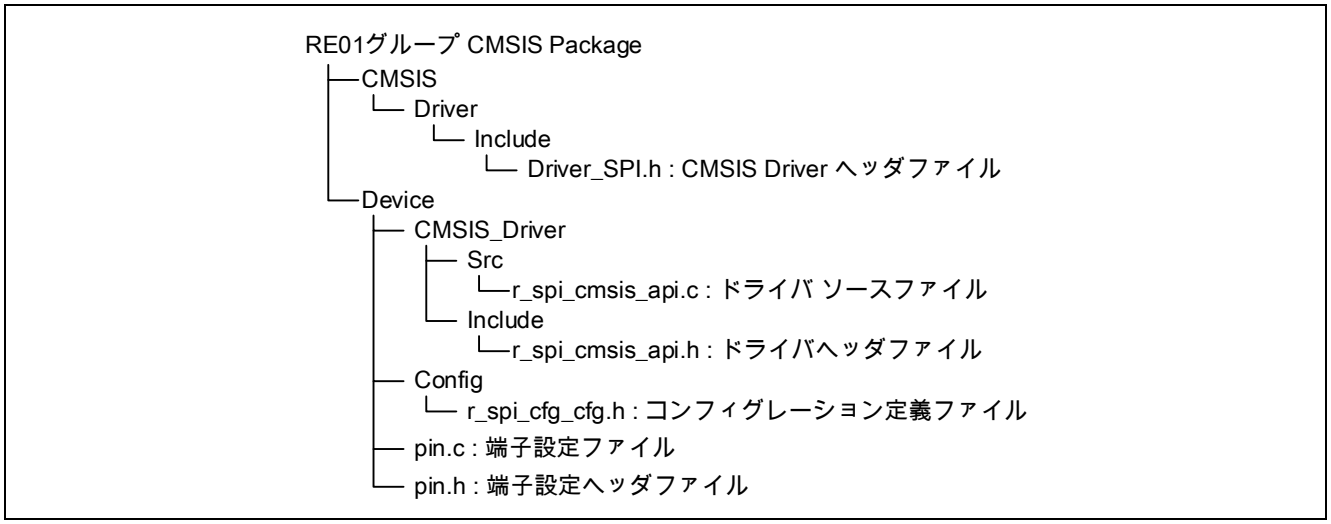


図 2-1 SPI ドライバファイル構成

2.2 ドライバ API

SPI 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。SPI 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、SPI ドライバへのアクセス例を図 2-3、図 2-4 に示します。

表 2-2 SPI ドライバのインスタンス一覧

インスタンス	内容
ARM_DRIVER_SPI Driver_SPI0	SPI0 を使用する場合のインスタンス
ARM_DRIVER_SPI Driver_SPI1	SPI1 を使用する場合のインスタンス

```
#include "R_Driver_SPI.h"

// SPI driver instance ( SPI0 )
extern ARM_DRIVER_SPI Driver_SPI0;
ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
```

図 2-2 SPI ドライバ インスタンス宣言例

表 2-3 SPI ドライバ API

API	内容	参照
Initialize	SPI ドライバの初期化（RAM の初期化、NVIC への割り込み登録、リソースロックの解除）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	SPI ドライバを解放（端子の解放、モジュールストップ状態への遷移）します モジュールストップ状態でない場合、モジュールスタート状態に遷移後レジスタの初期化と端子の解放を行った後、モジュールストップを行います 送受信に DMA を使用している場合、DMA ドライバの解放も行います	4.1.4
PowerControl	SPI 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
GetDataCount	送受信数を取得します	4.1.9
Control	SPI の制御コマンドを実行します 制御コマンドについては「2.5.1 SPI 制御コマンド定義」を参照	4.1.10
GetStatus	SPI の状態を取得します	4.1.11
GetVersion	SPI ドライバのバージョンを取得します	4.1.1
GetCapabilities	SPI ドライバの機能を取得します	4.1.2

表 2-4 SPI 制御コマンド定義（機能設定定義）一覧

コマンド	内容
ARM_SPI_MODE_INACTIVE	SPI を非アクティブ状態への遷移、端子設定を行います SPI レジスタの値が初期化されます
ARM_SPI_MODE_MASTER	マスタモードで SPI を初期化し、端子設定を行います SS 動作選択定義、フレームフォーマット定義、データビット長定義、ビットオーダー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_SPI_MODE_SLAVE	スレーブモードで SPI を初期化し、端子設定を行います SS 動作選択定義、フレームフォーマット定義、データビット長定義、ビットオーダー定義と組み合わせて指定してください
ARM_SPI_SET_BUS_SPEED	転送速度を設定します 第 2 引数にはボーレートを指定してください
ARM_SPI_GET_BUS_SPEED	転送速度を取得します
ARM_SPI_SET_DEFAULT_TX_VALUE	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_SPI_CONTROL_SS	SSL0 端子制御を行います 第 2 引数には以下の値を設定してください ARM_SPI_SS_INACTIVE：SSL0 端子非アクティブ("H") ARM_SPI_SS_ACTIVE：SSL0 端子アクティブ("L")
ARM_SPI_ABORT_TRANSFER	通信を中断します
ARM_SPI_MODE_MASTER_SIMPLEX	使用禁止(注)
ARM_SPI_MODE_SLAVE_SIMPLEX	使用禁止(注)

注　SPI 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。

マスタモード/スレーブモードで SPI 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）

マスタモードを SPI 動作（ハードウェア制御によるスレーブセレクト制御出力）、クロック極性：アイドル時の RSPCK が Low/クロック位相：立ち上がりエッジでデータサンプリング、立ち下がりエッジでデータ変化、8 ビットデータ長、MSB ファースト、100kbps で初期化する場合

```
spi0Drv -> Control(ARM_SPI_MODE_MASTER | ARM_SPI_SS_MASTER_HW_OUTPUT |
                   ARM_SPI_MSB_LSB | ARM_SPI_CPOL0_CPHA0 | ARM_SPI_DATA_BITS(8),   100000);
```

SPI で指定できる機能を表 2-5～表 2-8 に、各モードでの SPI アクセス例を図 2-3、図 2-4 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 SS 動作選択定義一覧

コマンド	内容
ARM_SPI_SS_MASTER_UNUSED（デフォルト）	マスタ動作時、SSL 信号を使用しません
ARM_SPI_SS_MASTER_SW	マスタ動作時、ソフトウェア制御によるスレーブセレクト制御を使用します
ARM_SPI_SS_MASTER_HW_OUTPUT	マスタ動作時、ハードウェア制御によるスレーブセレクト制御を出力します
ARM_SPI_SS_MASTER_HW_INPUT	使用禁止(注)
ARM_SPI_SS_SLAVE_HW	スレーブ動作時、ハードウェア制御でスレーブセレクト入力を監視します
ARM_SPI_SS_SLAVE_SW	スレーブ動作時、ソフトウェア制御でスレーブセレクト入力を監視します

注　SPI ドライバではサポートしていません。

表 2-6 SPI フレームフォーマット定義一覧

コマンド	内容
ARM_SPI_CPOL0_CPHA0（デフォルト）	クロック極性(0)：アイドル時の RSPCK が Low クロック位相(0)：立ち上がりエッジでデータサンプリング、立ち下がりエッジでデータ変化
ARM_SPI_CPOL0_CPHA1	クロック極性(0)：アイドル時の RSPCK が Low クロック位相(1)：立ち上がりエッジでデータ変化、立ち下がりエッジでデータサンプリング
ARM_SPI_CPOL1_CPHA0	クロック極性(1)：アイドル時の RSPCK が High クロック位相(0)：立ち上がりエッジでデータサンプリング、立ち下がりエッジでデータ変化
ARM_SPI_CPOL1_CPHA1	クロック極性(1)：アイドル時の RSPCK が High クロック位相(1)：立ち上がりエッジでデータ変化、立ち下がりエッジでデータサンプリング
ARM_SPI_TI_SSI	使用禁止(注)
ARM_SPI_MICROWIRE	使用禁止(注)

注　SPI ドライバではサポートしていません。

表 2-7 データビット長定義

コマンド	内容
ARM_SPI_DATA_BITS (n)	n に使用するデータビット長を指定してください データビット長には 8～16、20、24、32 ビットを指定できます

表 2-8 ビットオーダー定義一覧

コマンド	内容
ARM_SPI_MSB_LSB（デフォルト）	MSB ファースト
ARM_SPI_LSB_MSB	LSB ファースト

```
#include "Driver_SPI.h"

static void spi_callback (uint32_t event);

// SPI driver instance ( SPI0 )
extern ARM_DRIVER_SPI Driver_SPI0;
ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;

// Receive Buffer

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

    (void)spi0Drv->Initialize(spi_callback);                    /* SPI ドライバ初期化 */
    (void)spi0Drv->PowerControl(ARM_POWER_FULL);                /* SPI のモジュールストップ解除 */
    (void)spi0Drv->Control(ARM_SPI_MODE_MASTER |                /* マスタモード */
                        ARM_SPI_CPOL0_CPHA0 |                   /* クロック極性 0、クロック位相 0 */
                        ARM_SPI_LSB_MSB |                       /* LSB ファースト */
                        ARM_SPI_SS_MASTER_HW_OUTPUT |           /* HW 制御によるスレーブセレクト制御 */
                        ARM_SPI_DATA_BITS(8) ,                  /* 8 ビットデータ長 */
                        100000);                                /* 通信速度：100kbps */
    (void)spi0Drv-> Transfer (&tx_data[0], &rx_data[0], 3);     /* 3 バイト送受信開始 */

    while(1);
}

/**************************************************************************************************
* callback function
**************************************************************************************************/
static void spi_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_SPI_EVENT_TRANSFER_COMPLETE:
        {
            /* 正常に通信完了した場合の処理を記述 */
        }
        break;

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

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

図 2-3 SPI ドライバへのアクセス例（マスタモード）

```
#include "Driver_SPI.h"

static void spi_callback (uint32_t event);

// SPI driver instance ( SPI0 )
extern ARM_DRIVER_SPI Driver_SPI0;
ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;

// Receive Buffer

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

    (void)spi0Drv->Initialize(spi_callback);                    /* SPI ドライバ初期化 */
    (void)spi0Drv->PowerControl(ARM_POWER_FULL);                /* SPI のモジュールストップ解除 */
    (void)spi0Drv->Control(ARM_SPI_MODE_SLAVE |                 /* スレーブモード */
                        ARM_SPI_CPOL0_CPHA0 |                   /* クロック極性 0、クロック位相 0 */
                        ARM_SPI_LSB_MSB |                       /* LSB ファースト */
                        ARM_SPI_SS_SLAVE_HW |                   /* HW 制御によるスレーブセレクト入力を監視
*/
                        ARM_SPI_DATA_BITS(8) ,                  /* 8 ビットデータ長 */
                        0);                                     /* 通信速度：設定なし */
    (void)spi0Drv-> Transfer (&tx_data[0], &rx_data[0], 3);               /* 3 バイト送受信開始 */

    while(1);
}

/**************************************************************************************************
* callback function
**************************************************************************************************/
static void spi_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_SPI_EVENT_TRANSFER_COMPLETE:
        {
            /* 正常に通信完了した場合の処理を記述 */
        }
        break;

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

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

図 2-4 SPI ドライバへのアクセス例（スレーブモード）

2.3 端子設定

本ドライバで使用する端子は、pin.c の R_RSPI_Pinset_CHn(n=0,1)関数で設定、R_RSPI_Pinclr_CHn 関数で解放されます。R_RSPI_Pinset_CHn 関数は Control 関数でのマスタおよびスレーブモードの初期化、SPI 通信非アクティブ設定時に呼び出されます。R_RSPI_Pinclr_CHn 関数は PowerControl 関数でのパワーオフ設定時、または Uninitialize 関数で呼び出されます。

使用する端子は、pin.c の R_RSPI_Pinset_CHn、R_RSPI_Pinclr_CHn (n = 0,1)関数を編集して選択してください。端子設定の詳細は「5.2 端子設定について」を参照してください。

SPI 機能で使用される各端子名には_A、_B、_C および_D という接尾語が付加されています。SPI 機能を割り当てる場合、同じ接尾語の機能端子を選択してください。(注)

注　接尾語が同じ信号は、タイミング調整されているグループを表しています。違うグループの信号を同時に使用することはできません。例外として、SPI の「RSPCKA_C」と「MOSIA_C」は「_B」のグループと同時に使用します。また、「SSLB0_D」は「_B」のグループと同時に使用します。

2.4 通信制御および NVIC 割り込み設定

SPI 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_spi_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 2-9 に、送信/受信制御方法の定義を表 2-10 に示します。

表 2-9 送信/受信制御方法の設定定義（n = 0、1）

定義	初期値	内容
SPIn_TRANSMIT_CONTROL	SPI_USED_INTERRUPT	SPIn の送信制御（初期値：割り込み）
SPIn_RECEIVE_CONTROL	SPI_USED_INTERRUPT	SPIn の受信制御（初期値：割り込み）

表 2-10 送信/受信制御方法の定義

定義	値	内容
SPI_USED_INTERRUPT	(0)	送信/受信制御に割り込みを使用
SPI_USED_DMAC0	(1<<0)	送信/受信制御に DMAC0 を使用
SPI_USED_DMAC1	(1<<1)	送信/受信制御に DMAC1 を使用
SPI_USED_DMAC2	(1<<2)	送信/受信制御に DMAC2 を使用
SPI_USED_DMAC3	(1<<3)	送信/受信制御に DMAC3 を使用
SPI_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 2-11 に、NVIC への割り込み登録例を図 2-5 示します。

表 2-11　使用用途に対する NVIC の登録定義(n = 0,1、m = 0～3)

モード	使用用途	NVIC 登録定義
マスタ	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_SPIn_SPTI [送信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SPIn_SPII
	送受信、または 受信のみで使用(注)	[送信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SPIn_SPTI [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_SPIn_SPRI [受信制御に DMAC を使用時] なし
		SYSTEM_CFG_EVENT_NUMBER_SPIn_SPII SYSTEM_CFG_EVENT_NUMBER_SPIn_SPEI
スレーブ	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_SPIn_SPTI [送信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SPIn_SPTEND SYSTEM_CFG_EVENT_NUMBER_SPIn_SPEI
	送受信、または 受信のみで使用(注)	[送信制御に割り込み、DTC を使用時] SYSTEM_CFG_EVENT_NUMBER_SPIn_SPTI [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_SPIn_SPRI [受信制御に DMAC を使用時] SYSTEM_CFG_EVENT_NUMBER_DMACm_INT
		SYSTEM_CFG_EVENT_NUMBER_SPIn_SPEI

注　受信のみで使用する場合でもダミーデータの送信を行うため、送信側の設定も必要です。

```
・・・
#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_SPI0_SPRI
           (SYSTEM_IRQ_EVENT_NUMBER0)             /*!< Numbers 0/4/8/12/16/20/24/28 only */
#define SYSTEM_CFG_EVENT_NUMBER_SOL_DH
           (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)   /*!< Numbers 0/8/16/24 only */
・・・
#define SYSTEM_CFG_EVENT_NUMBER_SCI0_TXI
          (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)    /*!< Numbers 1/5/9/13/17/21/25/29 only */
#define SYSTEM_CFG_EVENT_NUMBER_SPI0_SPTI
          (SYSTEM_IRQ_EVENT_NUMBER1)              /*!< Numbers 1/5/9/13/17/21/25/29 only */
#define SYSTEM_CFG_EVENT_NUMBER_SOL_DL
          (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)    /*!< Numbers 1/9/17/25 only */
・・・
#define SYSTEM_CFG_EVENT_NUMBER_SCI0_TEI
          (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)    /*!< Numbers 2/6/10/14/18/22/26/30 only */
#define SYSTEM_CFG_EVENT_NUMBER_SPI0_SPII
          (SYSTEM_IRQ_EVENT_NUMBER2)              /*!< Numbers 2/6/10/14/18/22/26/30 only */
#define SYSTEM_CFG_EVENT_NUMBER_SPI0_SPTEND
          (SYSTEM_IRQ_EVENT_NUMBER6)              /*!< Numbers 2/6/10/14/18/22/26/30 only */
#define SYSTEM_CFG_EVENT_NUMBER_USBFS_USBI
          (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)    /*!< Numbers 2/10/18/26 only */
・・・
```

図 2-5 r_sysytem_cfg.h での NVIC への割り込み登録例(SPI0 使用時)

2.5 マクロ／型定義

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

2.5.1 SPI 制御コマンド定義

SPI 制御コマンドは、Control 関数の第 1 引数で使用する SPI 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、データビット長設定、SS 動作選択、ビットオーダー設定、フレームフォーマット設定の組み合わせで構成します。機能設定ビット（b0-b4）で SPI の通信モードを設定する場合は、データビット長設定、SS 動作選択、ビットオーダー設定、フレームフォーマット設定も設定してください。

SPI 制御コマンド定義の構成を図 2-6 に、各機能の設定定義を表 2-12～表 2-16 に示します。

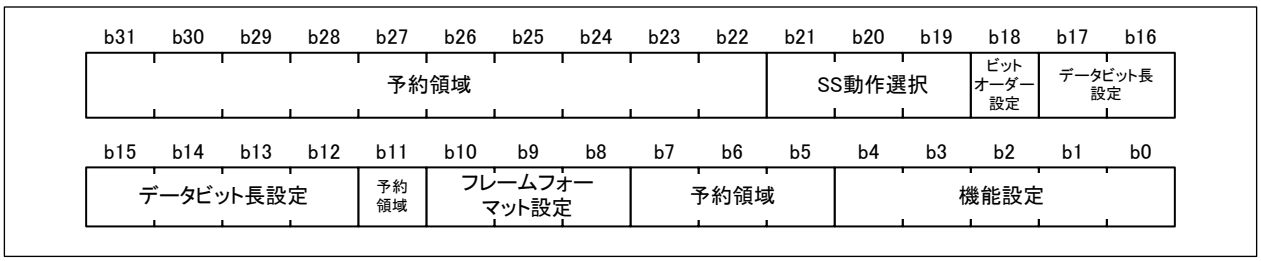


図 2-6 SPI 制御コマンド定義の構成

表 2-12 SPI 制御コマンド定義（機能設定定義）一覧

定義	値	内容
ARM_SPI_MODE_INACTIVE	(0x00UL << ARM_SPI_CONTROL_Pos)	SPI 非アクティブ設定
ARM_SPI_MODE_MASTER	(0x01UL << ARM_SPI_CONTROL_Pos)	SPI マスタモードで初期化
ARM_SPI_MODE_SLAVE	(0x02UL << ARM_SPI_CONTROL_Pos)	SPI スレーブモードで初期化
ARM_SPI_SET_BUS_SPEED	(0x10UL << ARM_SPI_CONTROL_Pos)	転送速度設定
ARM_SPI_GET_BUS_SPEED	(0x11UL << ARM_SPI_CONTROL_Pos)	転送速度取得
ARM_SPI_SET_DEFAULT_TX_VALUE	(0x12UL << ARM_SPI_CONTROL_Pos)	デフォルトデータ設定
ARM_SPI_CONTROL_SS	(0x13UL << ARM_SPI_CONTROL_Pos)	SSL0 端子制御設定
ARM_SPI_ABORT_TRANSFER	(0x14UL << ARM_SPI_CONTROL_Pos)	通信中断
ARM_SPI_MODE_MASTER_SIMPLEX	(0x03UL << ARM_SPI_CONTROL_Pos)	使用禁止(注)
ARM_SPI_MODE_SLAVE_SIMPLEX	(0x04UL << ARM_SPI_CONTROL_Pos)	使用禁止(注)

注　SPI 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_SPI_ERROR_MODE を返します。

表 2-13 SPI 制御コマンド（SS 動作選択定義）一覧

定義	値	内容
ARM_SPI_SS_MASTER_UNUSED	(0UL << ARM_SPI_SS_MASTER_MODE_Pos)	マスタ動作時、SSL 信号を使用しません
ARM_SPI_SS_MASTER_SW	(1UL << ARM_SPI_SS_MASTER_MODE_Pos)	マスタ動作時、ソフトウェア制御によるスレーブセレクト制御を使用します
ARM_SPI_SS_MASTER_HW_OUTPUT	(2UL << ARM_SPI_SS_MASTER_MODE_Pos)	マスタ動作時、ハードウェア制御によるスレーブセレクト制御を出力します
ARM_SPI_SS_MASTER_HW_INPUT	(3UL << ARM_SPI_SS_MASTER_MODE_Pos)	使用禁止(注)
ARM_SPI_SS_SLAVE_HW	(0UL << ARM_SPI_SS_SLAVE_MODE_Pos)	スレーブ動作時、ハードウェア制御によるスレーブセレクト入力を監視します
ARM_SPI_SS_SLAVE_SW	(1UL << ARM_SPI_SS_SLAVE_MODE_Pos)	スレーブ動作時、ソフトウェア制御によるスレーブセレクト入力を監視します

注　SPI 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_SPI_ERROR_SS_MODE を返します。

表 2-14 SPI 制御コマンド（フレームフォーマット定義）一覧

定義	値	内容
ARM_SPI_CPOL0_CPHA0	(0UL << ARM_SPI_FRAME_FORMAT_Pos)	クロック極性 0、クロック位相 0
ARM_SPI_CPOL0_CPHA1	(1UL << ARM_SPI_FRAME_FORMAT_Pos)	クロック極性 0、クロック位相 1
ARM_SPI_CPOL1_CPHA0	(2UL << ARM_SPI_FRAME_FORMAT_Pos)	クロック極性 1、クロック位相 0
ARM_SPI_CPOL1_CPHA1	(3UL << ARM_SPI_FRAME_FORMAT_Pos)	クロック極性 1、クロック位相 1
ARM_SPI_TI_SSI	(4UL << ARM_SPI_FRAME_FORMAT_Pos)	使用禁止(注)
ARM_SPI_MICROWIRE	(5UL << ARM_SPI_FRAME_FORMAT_Pos)	使用禁止(注)

注　SPI 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_SPI_ERROR_FRAME_FORMAT を返します。

表 2-15 SPI 制御コマンド（データビット長定義）一覧

定義	値	内容
ARM_SPI_DATA_BITS (n)	(((n) & 0x3F) << ARM_SPI_DATA_BITS_Pos)	データビット長設定 (n = 8～16、20、24、32)

注　SPI 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_SPI_ERROR_DATA_BITS を返します。

表 2-16 SPI 制御コマンド（ビットオーダー定義）一覧

定義	値	内容
ARM_SPI_MSB_LSB	(0UL << ARM_SPI_BIT_ORDER_Pos)	MSB ファースト
ARM_SPI_LSB_MSB	(1UL << ARM_SPI_BIT_ORDER_Pos)	LSB ファースト

2.5.2 SPI 特定のエラーコード定義

SPI 特定のエラーコード定義です。

表 2-17 SPI 特定エラーコード定義一覧

定義	値	内容
ARM_SPI_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_SPI_ERROR_FRAME_FORMAT	(ARM_DRIVER_ERROR_SPECIFIC - 2)	指定されたフレームフォーマットはサポートされていません
ARM_SPI_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_SPI_ERROR_BIT_ORDER	(ARM_DRIVER_ERROR_SPECIFIC - 4)	指定されたビットオーダーはサポートされていません
ARM_SPI_ERROR_SS_MODE	(ARM_DRIVER_ERROR_SPECIFIC - 5)	指定されたスレーブセレクトモードはサポートされていません

2.5.3 SSL 信号制御定義

Control 関数の ARM_SPI_CONTROL_SS コマンドで使用する SSL 信号制御用定義です。

表 2-18 SSL 信号制御定義一覧

定義	値	内容
ARM_SPI_SS_INACTIVE	(0)	SSL0 出力を非アクティブ(“H”)にします
ARM_SPI_SS_ACTIVE	(1)	SSL0 出力をアクティブ(“L”)にします

2.5.4 SPI 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。コールバック関数でのイベントコード判定例は、API アクセス例（「2.2 ドライバ API」の図 2-3、図 2-4）を参照してください。

表 2-19 SPI イベントコード一覧

定義	値	内容
ARM_SPI_EVENT_TRANSFER_COMPLETE	(1UL << 0)	通信が完了しました
ARM_SPI_EVENT_DATA_LOST	(1UL << 1)	オーバランエラーもしくはアンダランエラーにより送受信データが欠落しました
ARM_SPI_EVENT_MODE_FAULT(注)	(1UL << 2)	モードフォルトエラーが発生しました

注　本ドライバでは、発生しません。

2.6 構造体定義

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

2.6.1 ARM_SPI_STATUS 構造体

GetStatus 関数で SPI の状態を返すときに使用する構造体です。

表 2-20 ARM_SPI_STATUS 構造体

要素名	型	内容
busy	uint32_t:1	通信状態を示します 0: 通信待機中 1: 通信中(ビジー)
data_lost	uint32_t:1	オーバランエラーおよびアンダランエラーの発生状態を示します 0: オーバランエラーおよびアンダランエラー未発生 1: オーバランエラーおよびアンダランエラー発生
mode_fault	uint32_t:1	未使用(0 固定)
reserved	uint32_t:29	予約領域

2.6.2 ARM_SPI_CAPABILITIES 構造体

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

表 2-21 ARM_SPI_STATUS 構造体

要素名	型	内容	値
simplex	uint32_t:1	シンプレックスモード（マスタ/スレーブ）の有効/無効	0(無効)
ti_ssi	uint32_t:1	TI 同期シリアル・インターフェースの有効/無効	0(無効)
microwire	uint32_t:1	Microwire インターフェイスの有効/無効	0(無効)
event_mode_fault	uint32_t:1	信号モード障害イベント: ARM_SPI_EVENT_MODE_FAULT の有効/無効	0(無効)
reserved	uint32_t:29	予約領域	-

2.7 状態遷移

SPI ドライバの状態遷移図を図 2-7 に、各状態でのイベント動作を表 2-22 に示します。

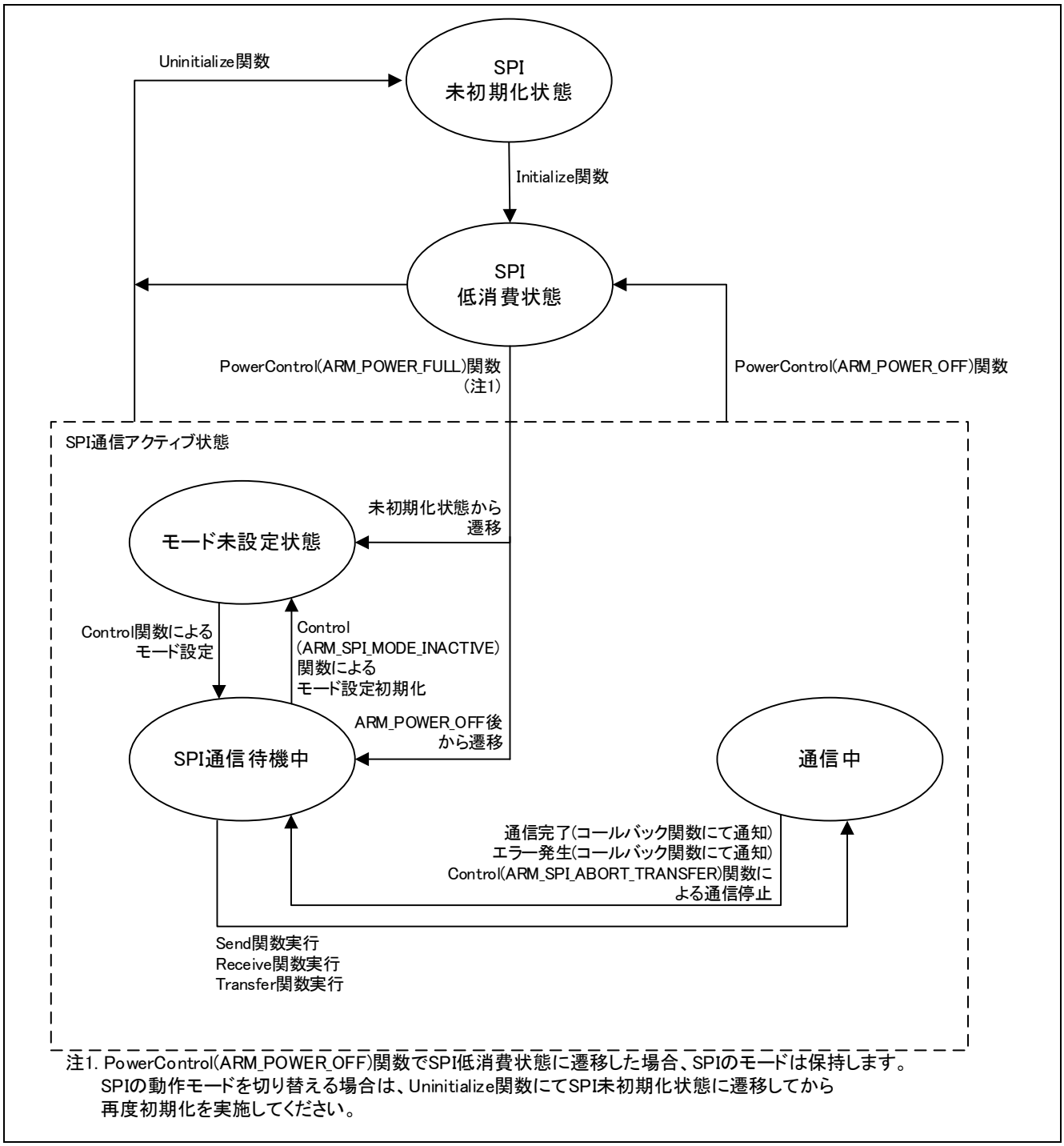


図 2-7 SPI ドライバの状態遷移

表 2-22 SPI 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
SPI 未初期化状態	リセット解除後の SPI ドライバの状態です	Initialize 関数の実行	SPI 低消費状態に遷移
SPI 低消費状態	SPI 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	SPI 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 SPI 通信待機中に遷移(注 2)
モード未設定状態	SPI モードが未設定の状態です	Control(ARM_SPI_MODE_XXX)関数の実行(注 3)	SPI 通信待機中に遷移
		Uninitialize 関数の実行	SPI 未初期化状態に遷移
		PowerControl(ARM_POWER_OFF)関数の実行	SPI 低消費状態に遷移
SPI 通信待機中	通信待ち状態です	Uninitialize 関数の実行	SPI 未初期化状態に遷移
		PowerControl(ARM_POWER_OFF)関数の実行	SPI 低消費状態に遷移
		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)
		Control(ARM_SPI_CONTROL_SS)関数の実行	SSL 端子制御(注 4)
		Control(ARM_SPI_MODE_INACTIVE)関数の実行	モード未設定状態に遷移
通信中	通信状態です	Uninitialize 関数の実行	SPI 未初期化状態に遷移
		PowerControl(ARM_POWER_OFF)関数の実行	SPI 低消費状態に遷移
		通信の完了	SPI 通信待機中に遷移し、コールバック関数を呼び出します(注 5)
		エラー発生	SPI 通信待機中に遷移し、コールバック関数を呼び出します(注 5)
		Control(ARM_SPI_ABORT_TRANSFER)関数の実行	通信を中断し、SPI 通信待機中に遷移します

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

注2. SPI 未初期化状態から SPI 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
MASTER：マスタモード
SLAVE　：スレーブモード

注4. r_spi_cfg.h で SSL 端子を設定し、かつモード設定時にソフトウェア制御によるスレーブセレクト制御使用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

SPI ドライバは SPI 動作(4 線式)、クロック同期式動作(3 線式)によるシリアル通信を実現します。SPI 動作(4 線式)のマスタモードでは SSL 信号出力をハードウェアで制御、スレーブモードでは SSL 信号入力をハードウェアで監視します。クロック同期式動作(3 線式)の場合、SSL 信号はソフトウェアで制御および監視します。

本章ではマスタ/スレーブモードで SPI ドライバを設定する手順について示します。

3.1 マスタモード

3.1.1 マスタモード初期設定手順

マスタ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。

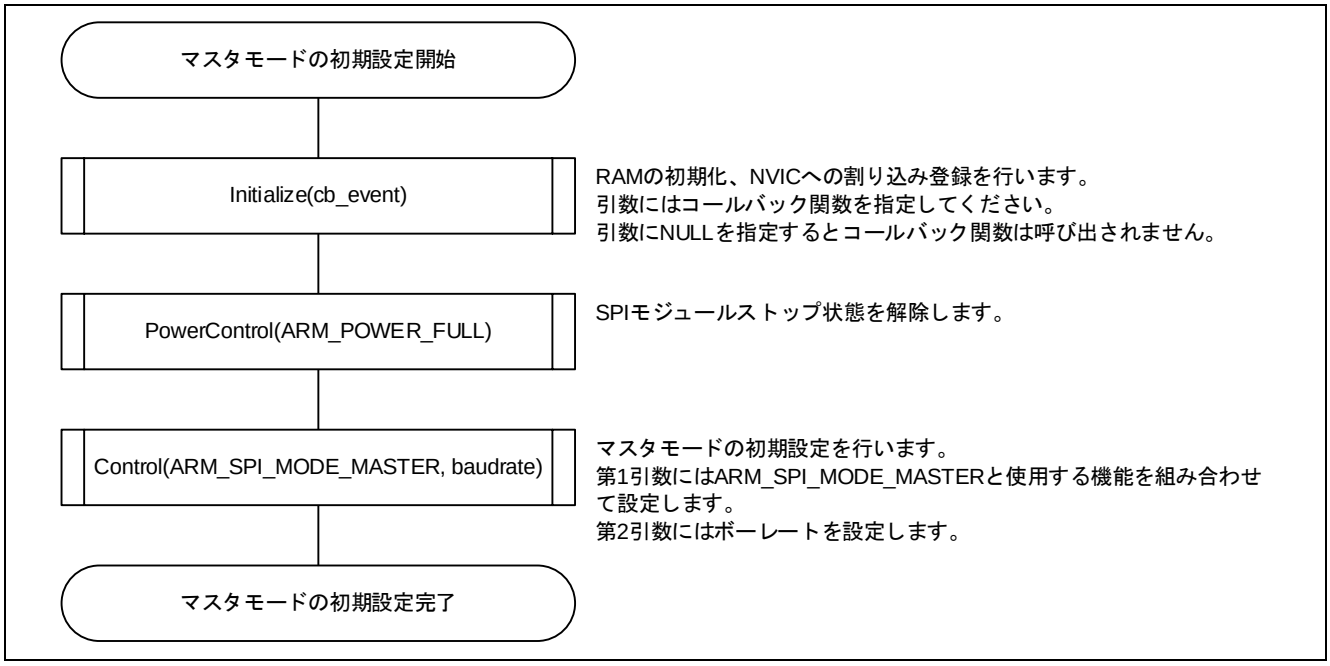


図 3-1 マスタモードの初期化手順

3.1.2 マスタモードでの送信処理

マスタモードで送信を行う手順を図 3-2 に示します。

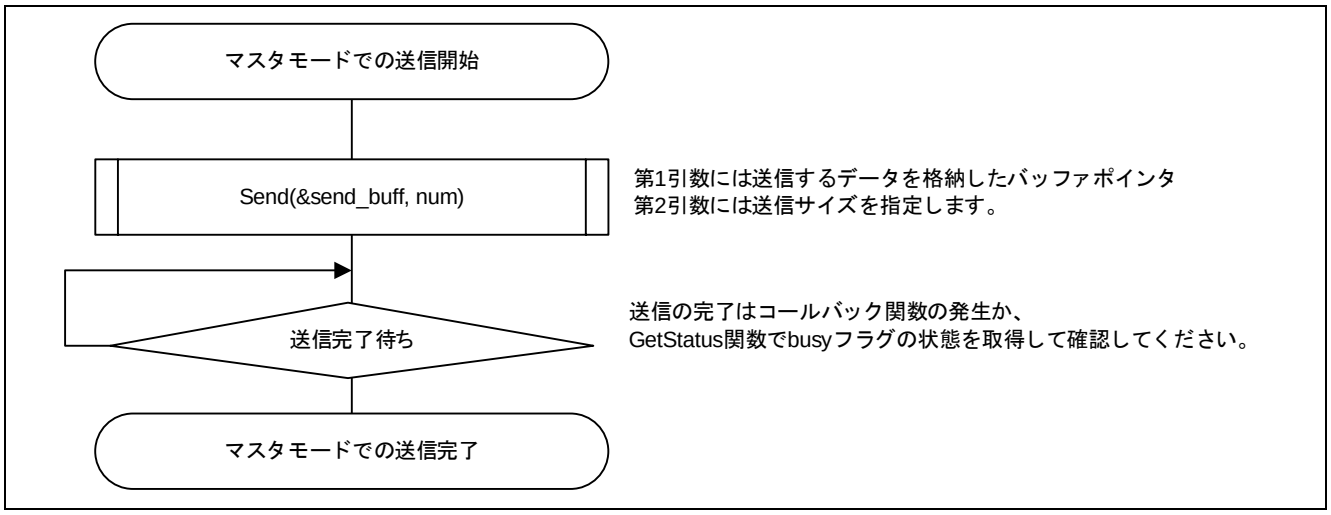
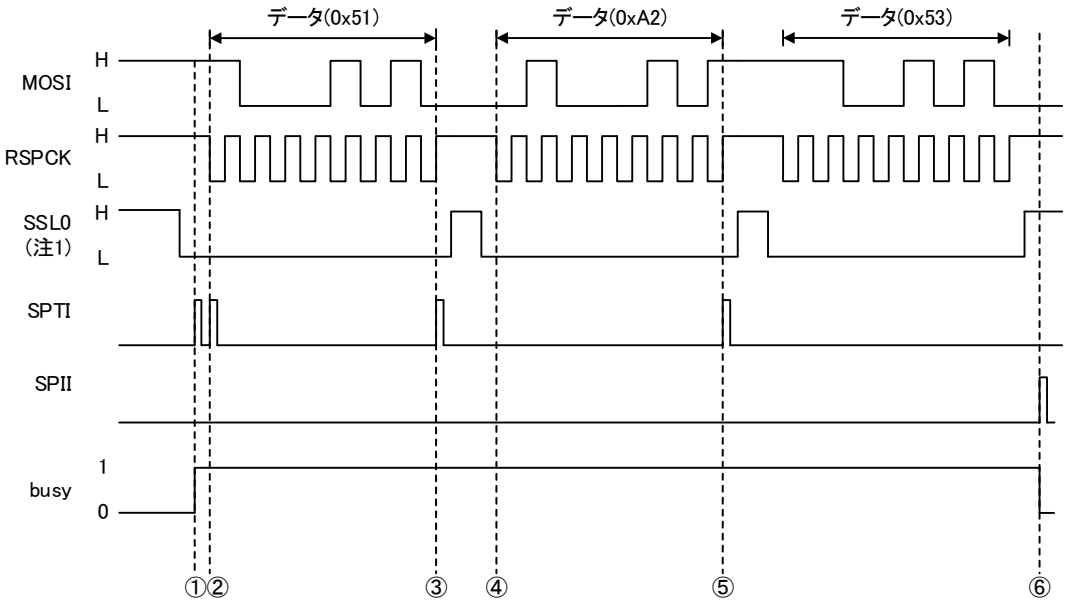


図 3-2 マスタモードでの送信手順

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_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数が呼び出されます。

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 3-3 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。



busy:通信状態(GetStatus関数で取得)

注1. HW制御によるSSL端子出力設定(ARM_SPI_SS_MASTER_HW_OUTPUT)時のみ出力されます

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、busy フラグが”1”(通信状態)になります。また、SPTI 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(SPDR)に書き込みます。

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

③ 3 回目の SPTI 割り込みにて最終送信データを SPDR に書き込みます。

④ ②で書き込んだ 2 バイト目のデータが出力されます。

⑤ ③で書き込んだ最終データの出力による SPTI 割り込みで、SPTI 割り込みを禁止にし、SPII 割り込みを許可にします。

⑥ 全データの送信完了後、SPII 割り込みが発生し、busy フラグが”0”（通信待ち状態）になります。SPII 割り込みでは、SPI 制御で使用するすべての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_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。

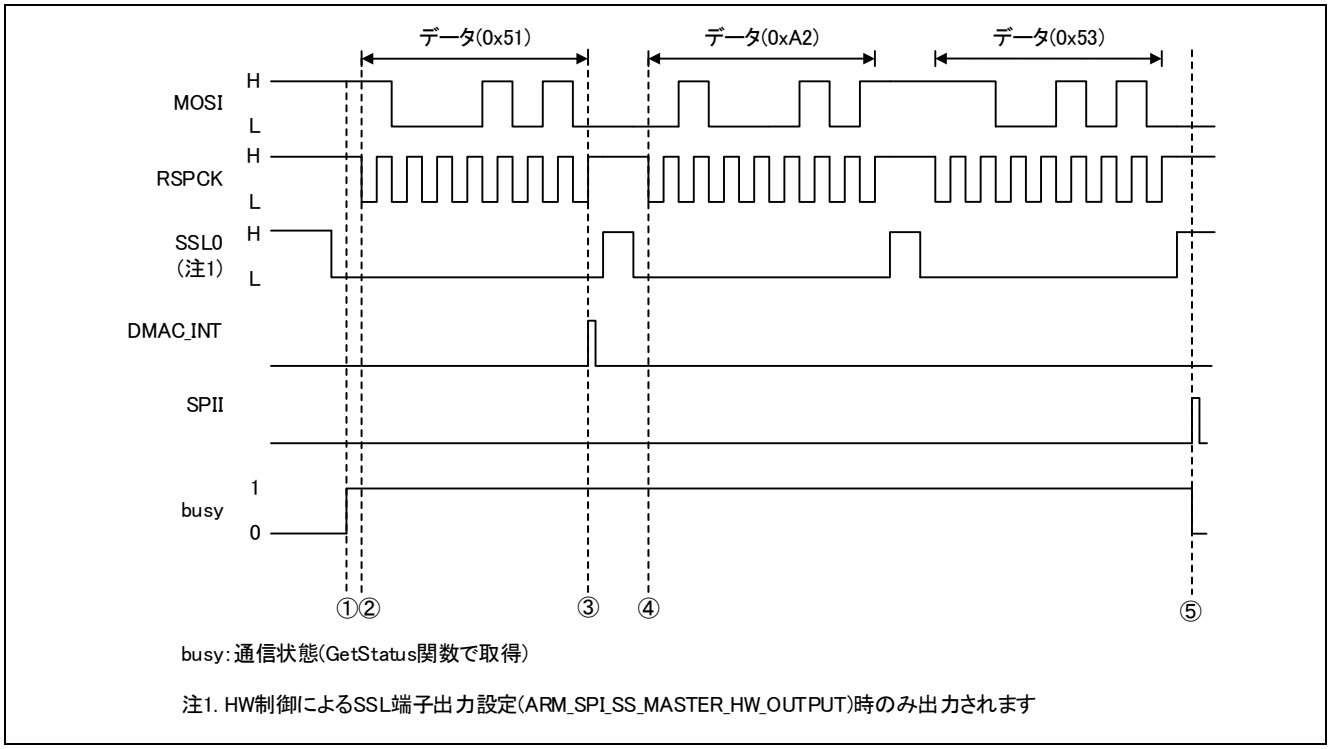


図 3-4 DMAC 制御による送信動作（3 バイト送信）

① Send 関数を実行すると、busy フラグを”1”(通信状態)に設定し、DMAC 転送要因に SPTI 割り込みを設定します。また、DMA 転送にて 1 バイト目の送信データが SPDR に転送されます。

② DMA 転送にて 2 バイト目の送信データが SPDR に転送されます。

③ DMA 転送にて最終データが転送されます。DMAC 転送完了割り込みが発生し、DMAC 転送完了割り込みを禁止、SPII 割り込みを許可にします。

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。

⑤ 全データの送信完了後、SPII 割り込みが発生し、busy フラグが”0”（通信待ち状態）になります。SPII 割り込みでは、SPI 制御で使用するすべての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_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。

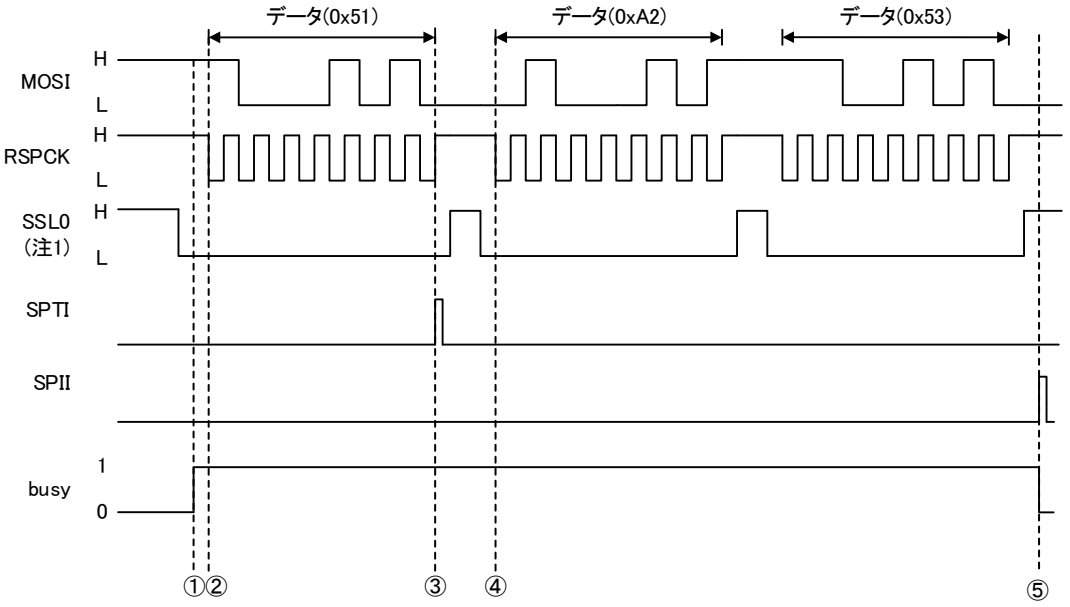


図 3-5 DTC 制御による送信動作（3 バイト送信）

① Send 関数を実行すると、busy フラグを”1”(通信状態)に設定、DTC 転送要因に SPTI 割り込みを設定します。また、DMA 転送にて 1 バイト目の送信データが SPDR に転送されます。

② DMA 転送にて 2 バイト目の送信データが SPDR に転送されます。

③ DMA 転送にて最終データが転送されます。SPTI 割り込みが発生し、SPTI 割り込みを禁止、SPII 割り込みを許可にします。

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。

⑤ 全データの送信完了後、SPII 割り込みが発生し、busy フラグが”0”（通信待ち状態）になります。SPII 割り込みでは、SPI 制御で使用するすべての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_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。

3.1.3 マスタモードでの受信処理

マスタモードで受信を行う手順を図 3-6 に示します。

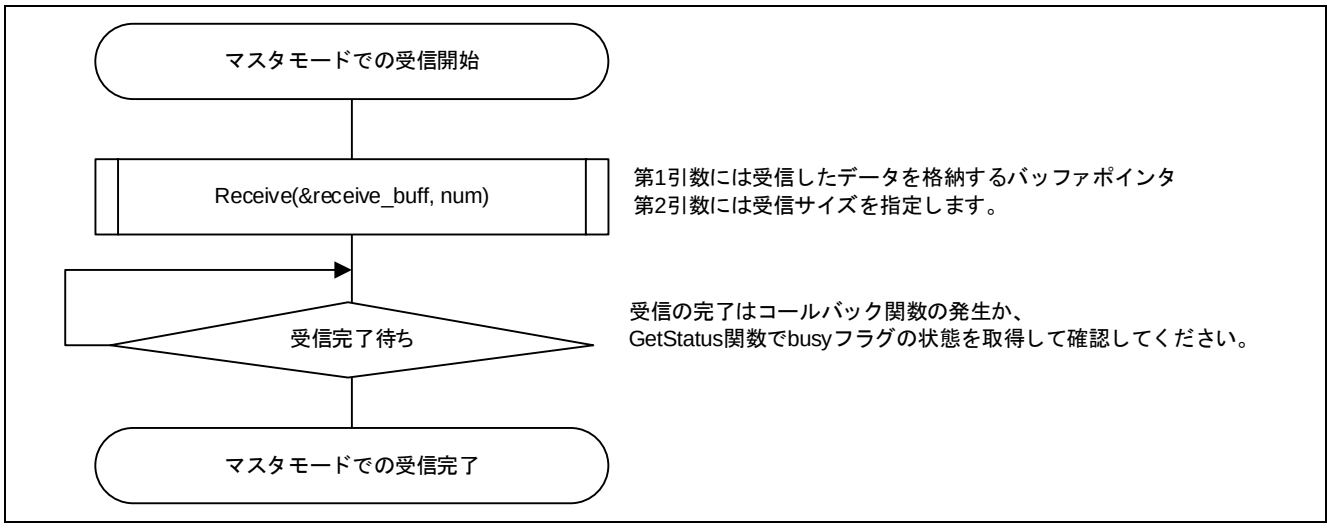


図 3-6 マスタモードでの受信手順

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_SPI_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、ARM_SPI_EVENT_DATA_LOST 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。

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、SPI 通信では受信のみの動作ができないため、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_SPI_SET_DEFAULT_TX_VALUE コマンドにて変更できます。

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。

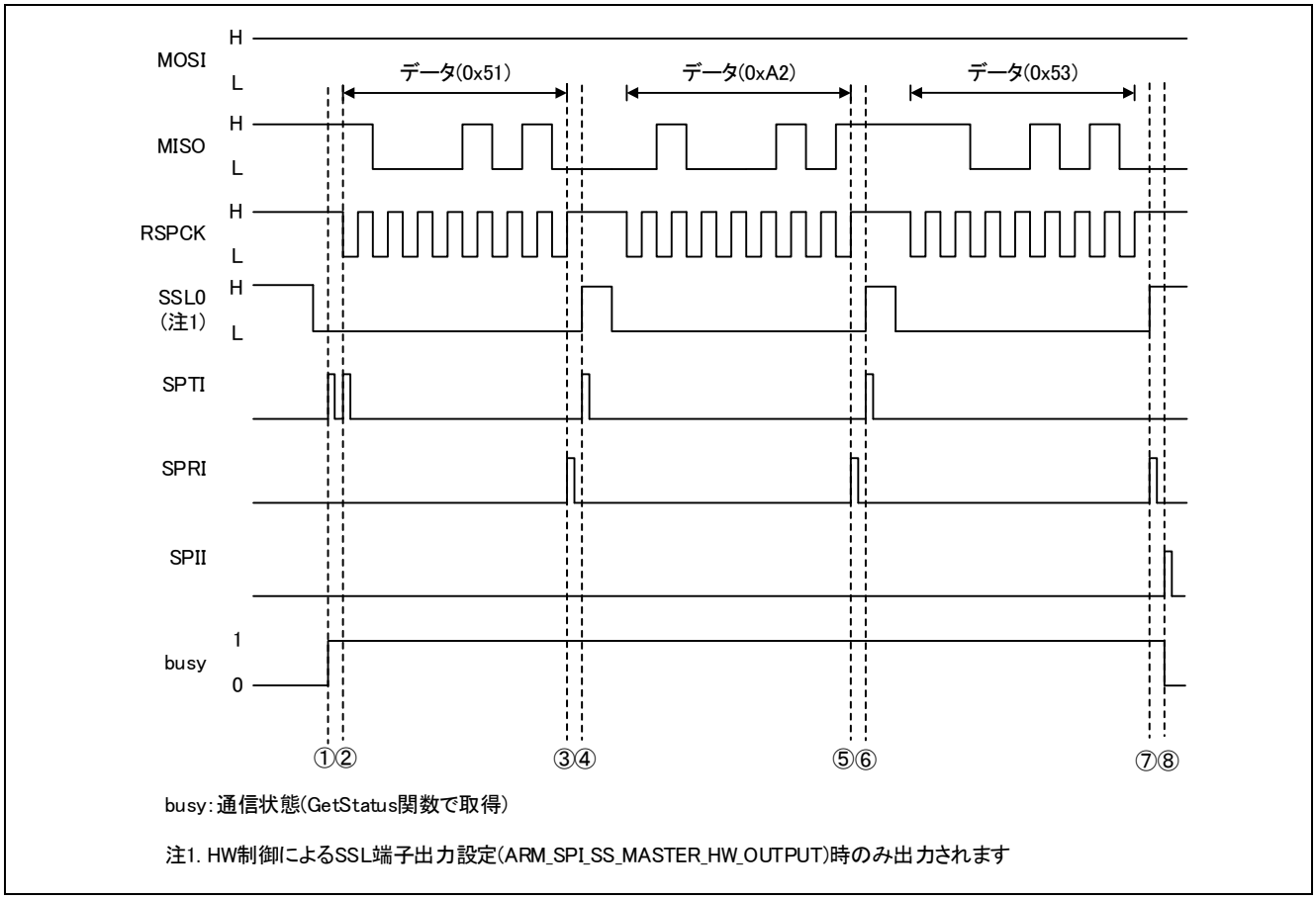


図 3-7 割り込み制御による受信動作（3 バイト受信、ダミーデータ 0xFF）

① Receive 関数を実行すると、busy フラグが”1”(通信状態)になります。また、SPTI 割り込みが発生し、ダミーデータをデータレジスタ(SPDR)に書き込みます。

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

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

④ 3 回目の SPTI 割り込みにて最終ダミーデータを SPDR に書き込みます。

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

⑥ ④で書き込んだ最終ダミーデータの出力による SPTI 割り込みで、SPTI 割り込みを禁止にし、SPII 割り込みを許可にします。

⑦ 最終データ読み出し時の SPRI 割り込みで、SPRI 割り込みを禁止にします。

⑧ 全データの受信完了後、SPII 割り込みが発生し、busy フラグが”0”（通信待ち状態）になります。SPII 割り込みでは、SPI 制御で使用するすべての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_SPI_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、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

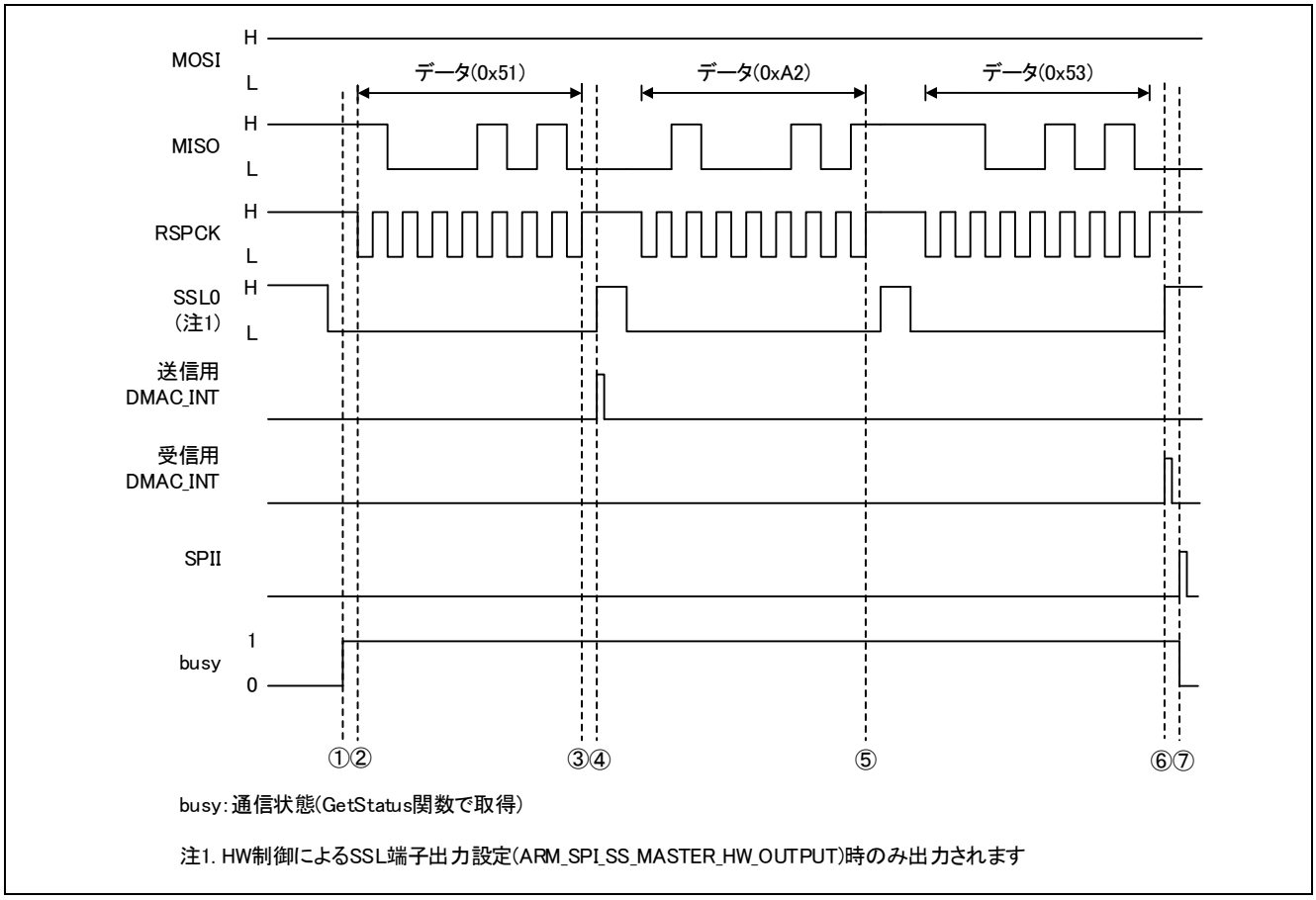


図 3-8 DMAC 制御による受信動作（3 バイト受信、ダミーデータ 0xFF）

① Receive 関数を実行すると、busy フラグを”1”(通信状態)に設定、DMAC 転送要因に SPTI 割り込み、SPRI 割り込みを設定します。また、DMA 転送にてダミーデータがデータレジスタ(SPDR)に転送されます。

② DMA 転送にて 2 バイト目のダミーデータを SPDR レジスタに転送します。

③ 1 バイト受信が完了すると、DMA 転送にてデータレジスタ(SPDR)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 転送完了割り込みにて、送信側の DMAC 転送完了割り込みを禁止にし、SPII 割り込みを許可にします。

⑤ 1 バイト受信完了ごとに DMA 転送にて、SPDR レジスタの受信データを指定されたバッファに転送します。

⑥ 最終データ受信完了時、受信側の DMAC 転送完了割り込みが発生します。受信側の DMA 転送完了割り込み処理にて受信側の DMAC 転送完了割り込みを禁止にします。

⑦ 全データの受信完了後、SPII 割り込みが発生し、busy フラグが”0”（通信待ち状態）になります。SPII 割り込みでは、SPI 制御で使用するすべての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_SPI_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、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

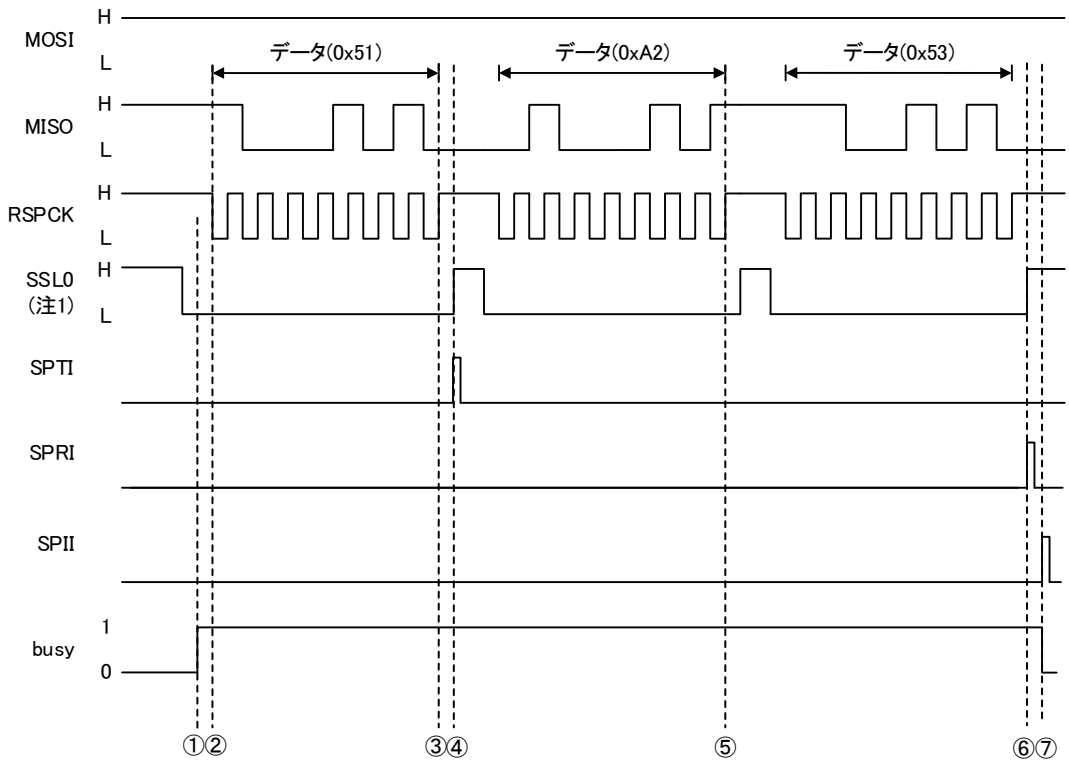


図 3-9 DTC 制御による受信動作（3 バイト受信、ダミーデータ 0xFF）

① Receive 関数を実行すると、busy フラグを”1”(通信状態)に設定、DTC 転送要因に SPTI 割り込み、SPRI 割り込みを設定します。また、DMA 転送にてダミーデータがデータレジスタ(SPDR)に転送されます。

② DMA 転送にて 2 バイト目のダミーデータを SPDR レジスタに転送します。

③ 1 バイト受信が完了すると、DMA 転送にてデータレジスタ(SPDR)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時の SPTI 割り込みにて、SPTI 割り込みを禁止、SPII 割り込みを許可にします。

⑤ 1 バイト受信完了ごとに DMA 転送にて、SPDR レジスタの受信データを指定されたバッファに転送します。

⑥ 最終データ受信完了時の SPRI 割り込みで、SPRI 割り込みを禁止にします。

⑦ 全データの受信完了後、SPII 割り込みが発生し、busy フラグが”0”（通信待ち状態）になります。SPII 割り込みでは、SPI 制御で使用するすべての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_SPI_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、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

3.1.4 マスタモードでの送受信処理

マスタモードで送受信を行う手順を図 3-10 に示します。

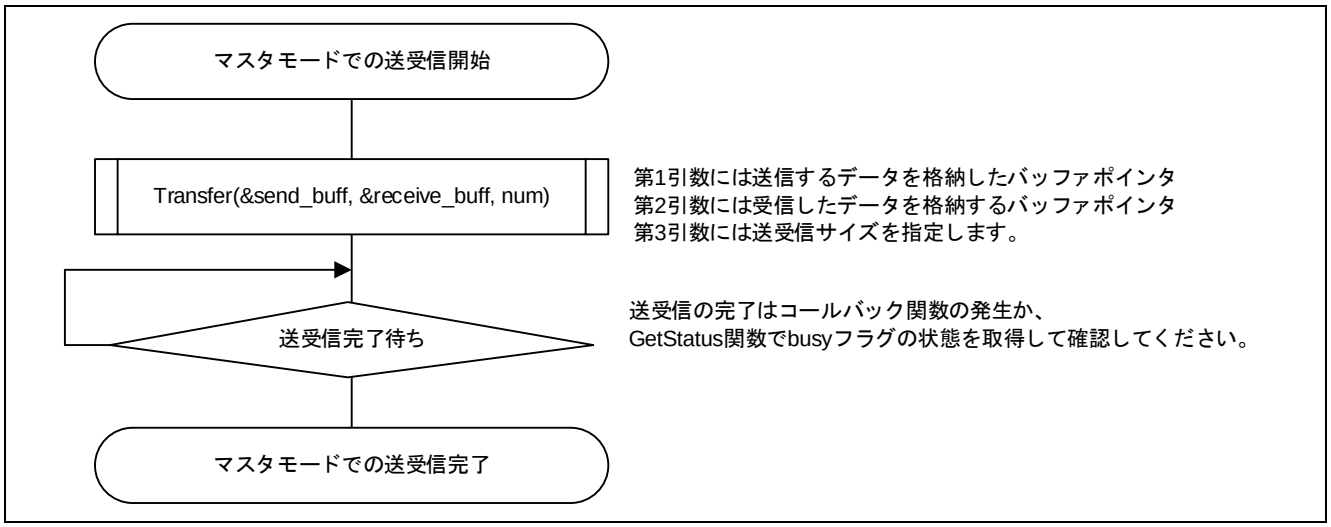


図 3-10 マスタモードでの送受信手順

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_SPI_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、ARM_SPI_EVENT_DATA_LOST 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。

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 3-11 に通信制御が割り込みの場合の動作を、図 3-12 に通信制御が DMAC の場合の動作を、図 3-13 に通信制御が DTC の場合の動作を示します。

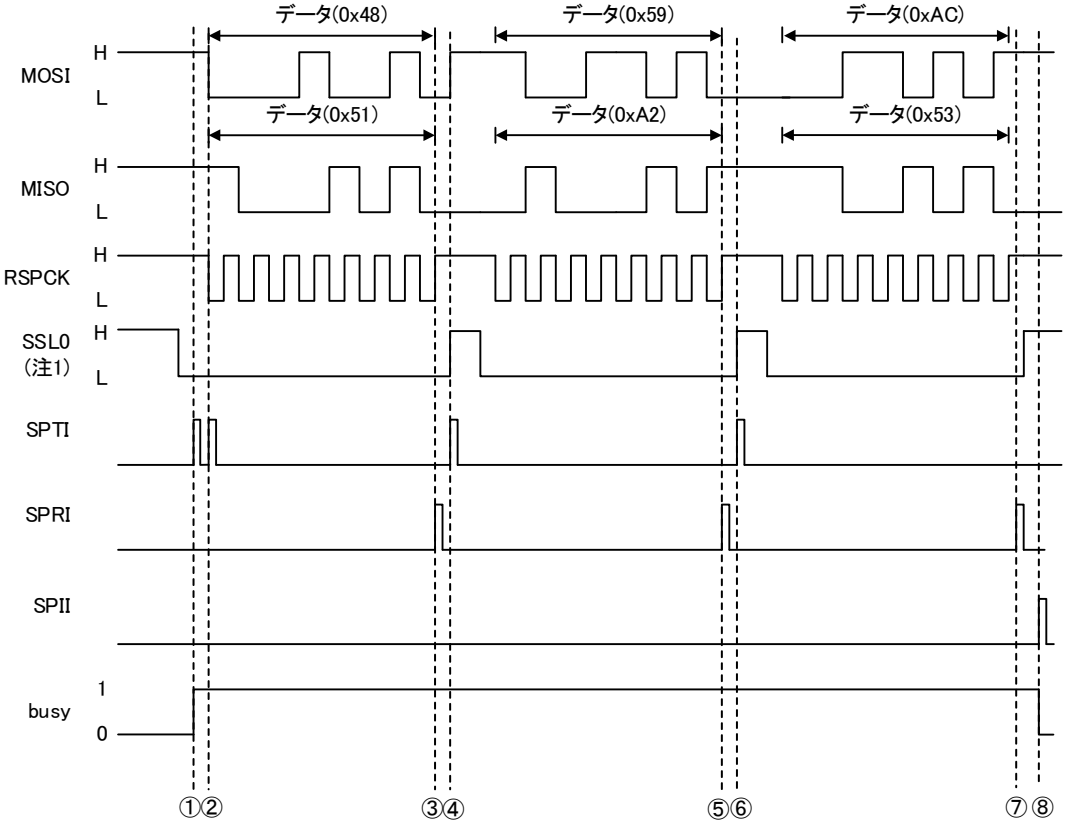


図 3-11 割り込み制御による送受信動作（3 バイト受信）

① Transfer 関数を実行すると、busy フラグが”1”(通信状態)になります。また、SPTI 割り込みが発生し、送信データをデータレジスタ(SPDR)に書き込みます。

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

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

④ 3 回目の SPTI 割り込みにて最終送信データを SPDR に書き込みます。

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

⑥ ④で書き込んだ最終データの出力による SPTI 割り込みで、SPTI 割り込みを禁止にし、SPII 割り込みを許可にします。

⑦ 最終データ読み出し時の SPRI 割り込みで、SPRI 割り込みを禁止にします。

⑧ 受信完了後、SPII 割り込みが発生し、busy フラグを”0”（通信待ち状態）にします。SPII 割り込みでは、SPI 制御で使用するすべての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_SPI_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、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

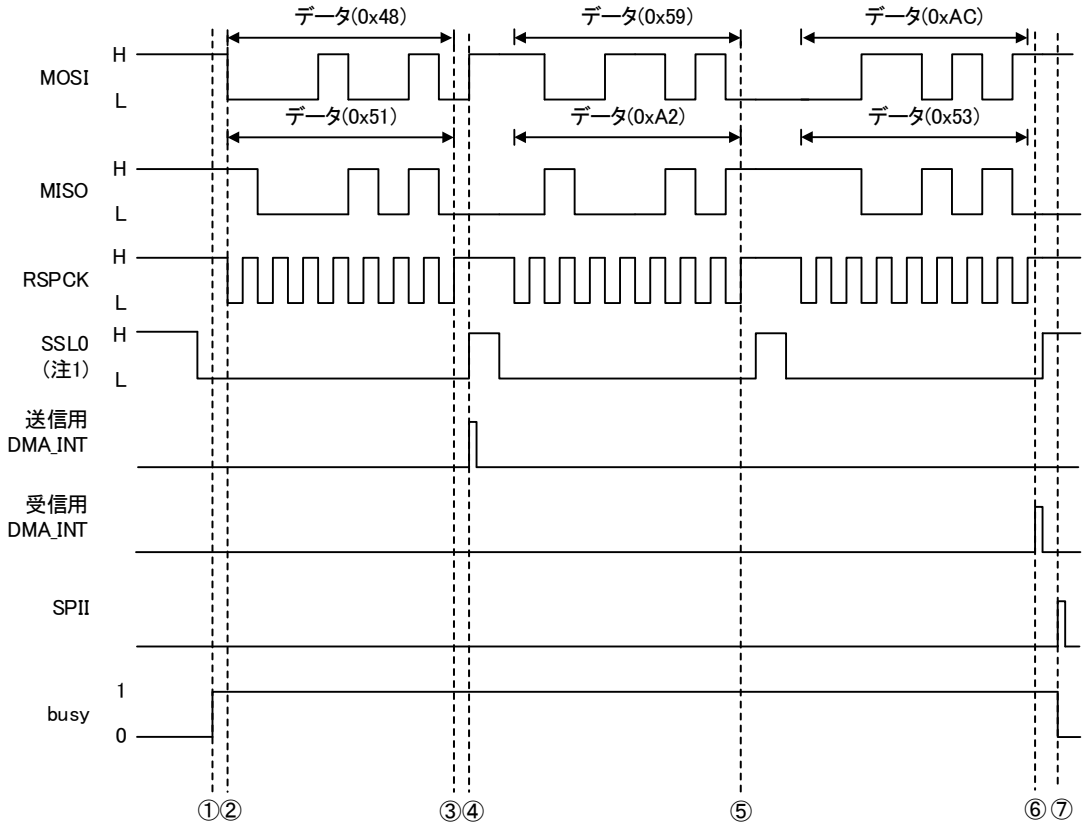


図 3-12 DMAC 制御による送受信動作（3 バイト受信）

① Transfer 関数を実行すると、busy フラグを”1”(通信状態)に設定、DMAC 転送要因に SPTI 割り込み、SPRI 割り込みを設定します。また、DMA 転送にて送信データがデータレジスタ(SPDR)に転送されます。

② DMA 転送にて 2 バイト目の送信データを SPDR レジスタに転送します。

③ 1 バイト受信が完了すると、DMA 転送にてデータレジスタ(SPDR)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 転送完了割り込みにて、送信側の DMAC 転送完了割り込みを禁止にし、SPII 割り込みを許可にします。

⑤ 1 バイト受信完了ごとに DMA 転送にて、SPDR レジスタの受信データを指定されたバッファに転送します。

⑥ 最終データ受信完了時の受信側の DMAC 転送完了割り込みが発生します。割り込み処理にて受信側の DMAC 転送完了割り込みを禁止にします。

⑦ 受信完了後、SPII 割り込みが発生し、busy フラグを”0”（通信待ち状態）にします。SPII 割り込みでは、SPI 制御で使用するすべての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_SPI_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、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

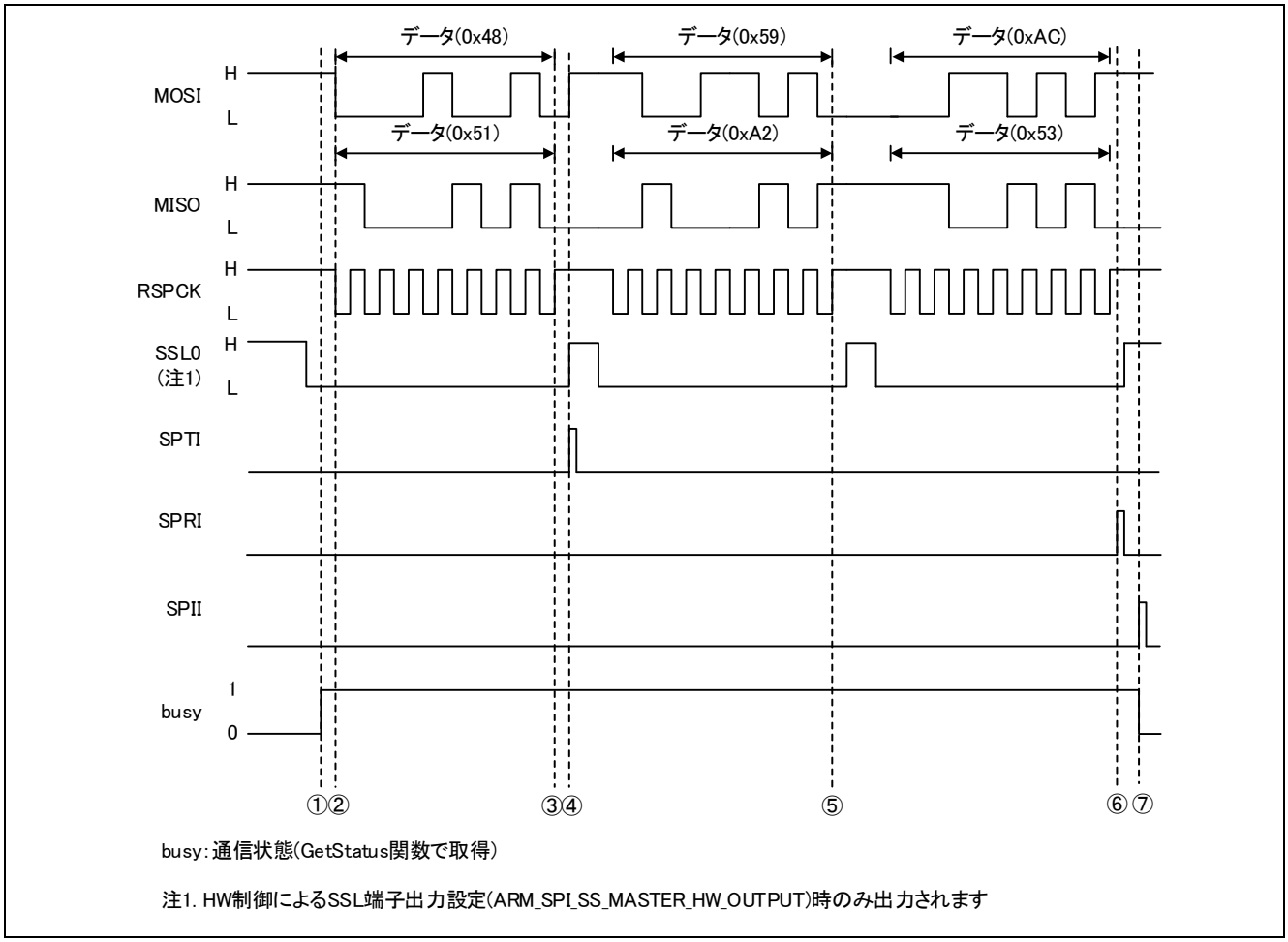


図 3-13 DTC 制御による送受信動作（3 バイト受信）

① Transfer 関数を実行すると、busy フラグを”1”(通信状態)に設定、DTC 転送要因に SPTI 割り込み、SPRI 割り込みを設定します。また、DMA 転送にて送信データがデータレジスタ(SPDR)に転送されます。

② DMA 転送にて 2 バイト目の送信データを SPDR レジスタに転送します。

③ 1 バイト受信が完了すると、DMA 転送にてデータレジスタ(SPDR)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時の SPTI 割り込みにて、SPTI 割り込みを禁止、SPII 割り込みを許可にします。

⑤ 1 バイト受信完了ごとに DMA 転送にて、SPDR レジスタの受信データを指定されたバッファに転送します。

⑥ 最終データ受信完了時の SPRI 割り込みで、SPRI 割り込みを禁止にします。

⑦ 受信完了後、SPII 割り込みが発生し、busy フラグを”0”（通信待ち状態）にします。SPII 割り込みでは、SPI 制御で使用するすべての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_SPI_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、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

3.2 スレーブモード

3.2.1 スレーブモード初期設定手順

スレーブモードの初期設定手順を図 3-14 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。

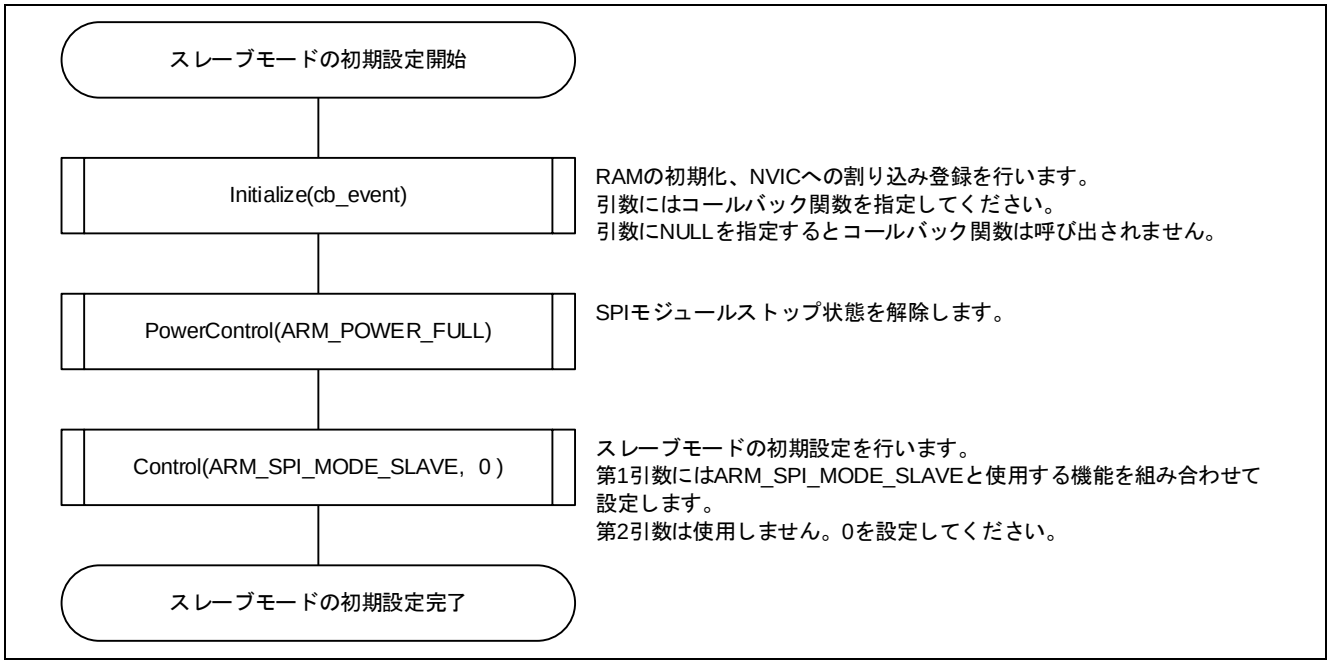


図 3-14 スレーブモードの初期化手順

3.2.2 スレーブモードでの送信処理

スレーブモードで送信を行う手順を図 3-15 に示します。

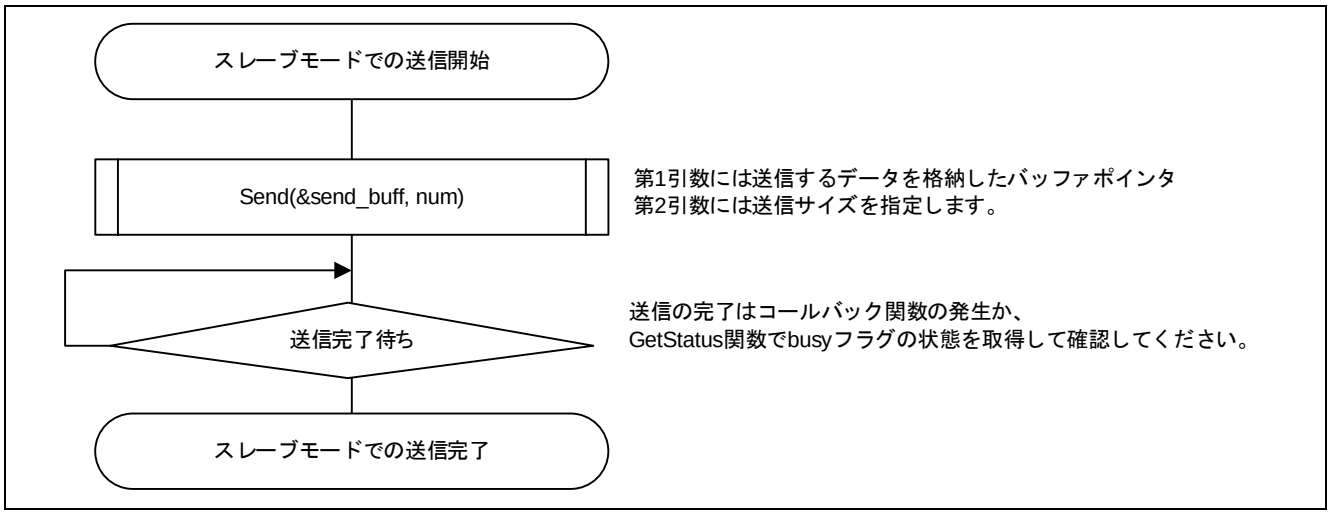


図 3-15 スレーブモードでの送信手順

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_SPI_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、送信エラーが発生した場合、ARM_SPI_EVENT_DATA_LOST 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。

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 3-16 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。

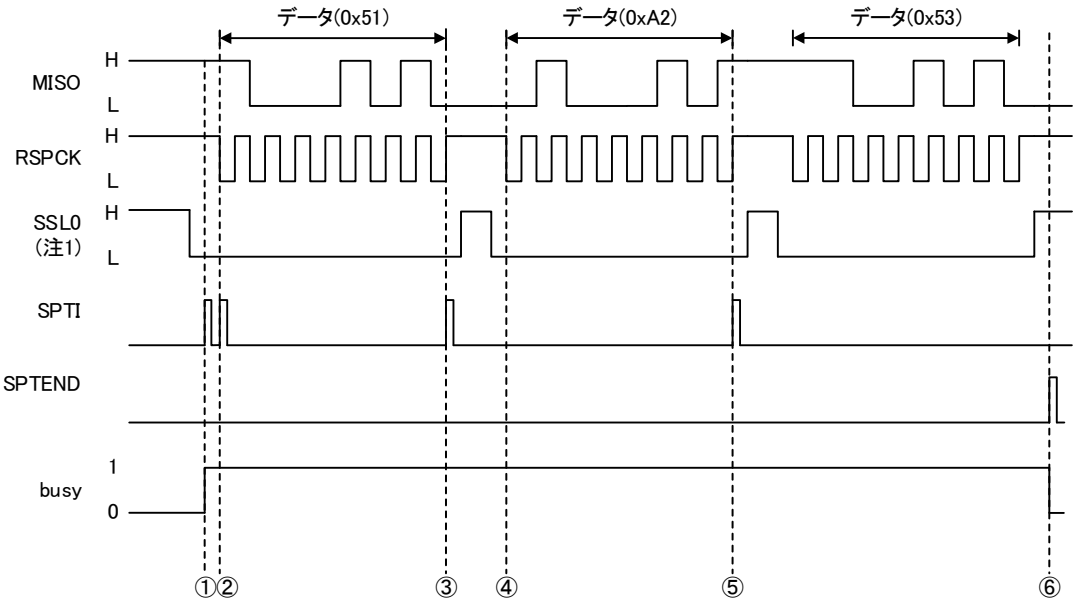


図 3-16 割り込み制御による送信動作（3 バイト送信）

① Send 関数を実行すると、busy フラグが”1”（通信ビジー）になります。また、SPTI 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(SPDR)に書き込みます。

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

③ 3 回目の SPTI 割り込みにて最終送信データを SPDR に書き込みます。

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。

⑤ 最終データ書き込み後の SPTI 割り込みで、SPTI 割り込みを禁止にし、SPTEND 割り込みを許可にします。

⑥ 送信が完了すると、SPTEND 割り込みが発生し、busy フラグが”0”（通信待ち状態）になります。SPTEND 割り込みでは、SPI 制御で使用するすべての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_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注 1)(注 2)

注1. 送信エラーが発生した場合、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

注2. CPHA = 0 設定時、通信を再開する場合は RSPCK の半サイクルをソフトウェアで待ってから実行してください。詳細は「5.6 スレーブモードかつ CPHA0 での通信再開について」参照。

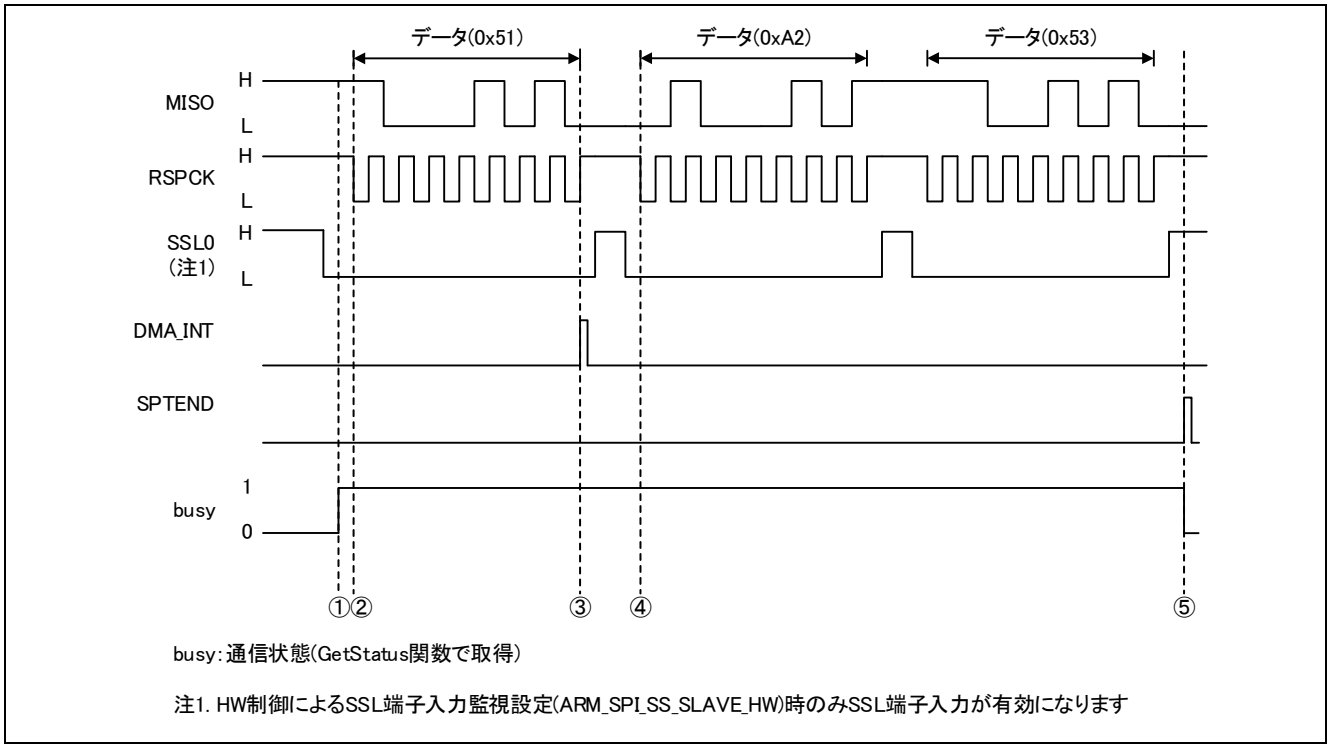


図 3-17 DMAC 制御による送信動作（3 バイト送信）

① Send 関数を実行すると、busy フラグを”1”(通信状態)に設定、DMAC 転送要因に SPTI 割り込みを設定します。また、DMA 転送にて 1 バイト目の送信データが SPDR に転送されます。

② RSPCK 端子にクロックが入力されると 1 バイト目のデータが MISO 端子から出力を開始、2 回目の DMA 転送にて 2 バイト目の送信データが SPDR レジスタに転送されます。

③ 3 回目の DMA 転送にて最終送信データを SPDR に書き込みます。DMAC 転送完了割り込みが発生し、DMAC 転送完了割り込みを禁止、SPTEND 割り込みを許可にします。

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。

⑤ 送信が完了すると、SPTEND 割り込みが発生し、busy フラグが”0”（通信待ち状態）になります。SPTEND 割り込みでは、SPI 制御で使用するすべての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_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注 1)(注 2)

注1. 送信エラーが発生した場合、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

注2. CPHA = 0 設定時、通信を再開する場合は RSPCK の半サイクルをソフトウェアで待ってから実行してください。詳細は「5.6 スレーブモードかつ CPHA0 での通信再開について」参照。

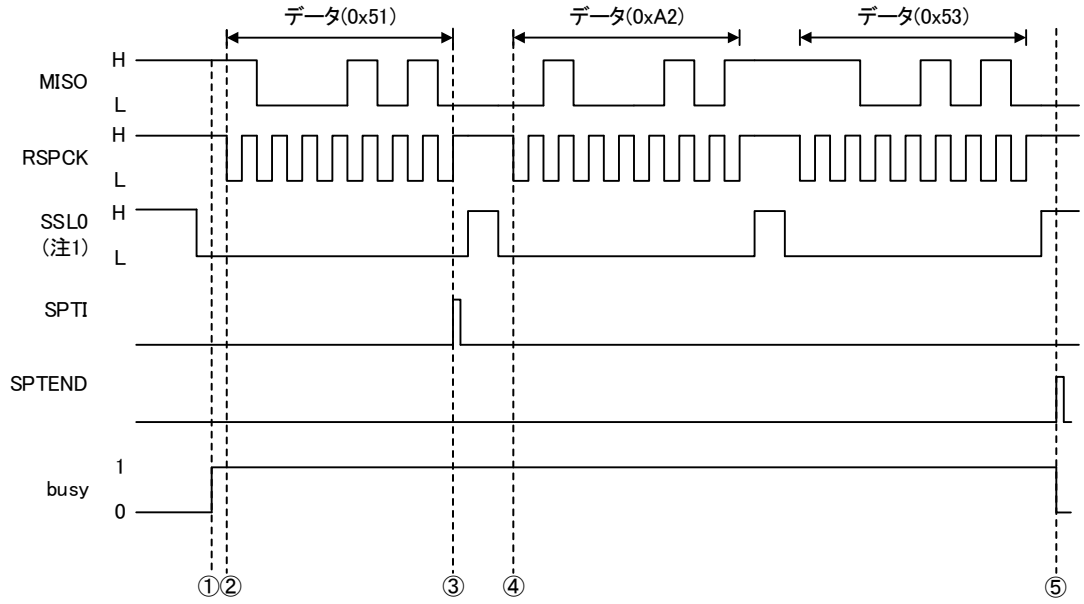


図 3-18 DTC 制御による送信動作（3 バイト送信）

① Send 関数を実行すると、busy フラグを”1”(通信状態)に設定、DTC 転送要因に SPTI 割り込みを設定します。また、DMA 転送にて 1 バイト目の送信データが SPDR に転送されます。

② RSPCK 端子にクロックが入力されると 1 バイト目のデータが MISO 端子から出力を開始、2 回目の DMA 転送にて 2 バイト目の送信データが SPDR レジスタに転送されます。

③ 3 回目の DMA 転送にて最終送信データを SPDR に書き込みます。最終データ書き込み後の SPTI 割り込みで、SPTI 割り込みを禁止、SPTEND 割り込みを許可にします。

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。

⑤ 送信が完了すると、SPTEND 割り込みが発生し、busy フラグが”0”（通信待ち状態）になります。SPTEND 割り込みでは、SPI 制御で使用するすべての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_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注 1)(注 2)

注1. 送信エラーが発生した場合、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

注2. CPHA = 0 設定時、通信を再開する場合は RSPCK の半サイクルをソフトウェアで待ってから実行してください。詳細は「5.6 スレーブモードかつ CPHA0 での通信再開について」参照。

3.2.3 スレーブモードでの受信処理

スレーブモードで受信を行う手順を図 3-19 に示します。

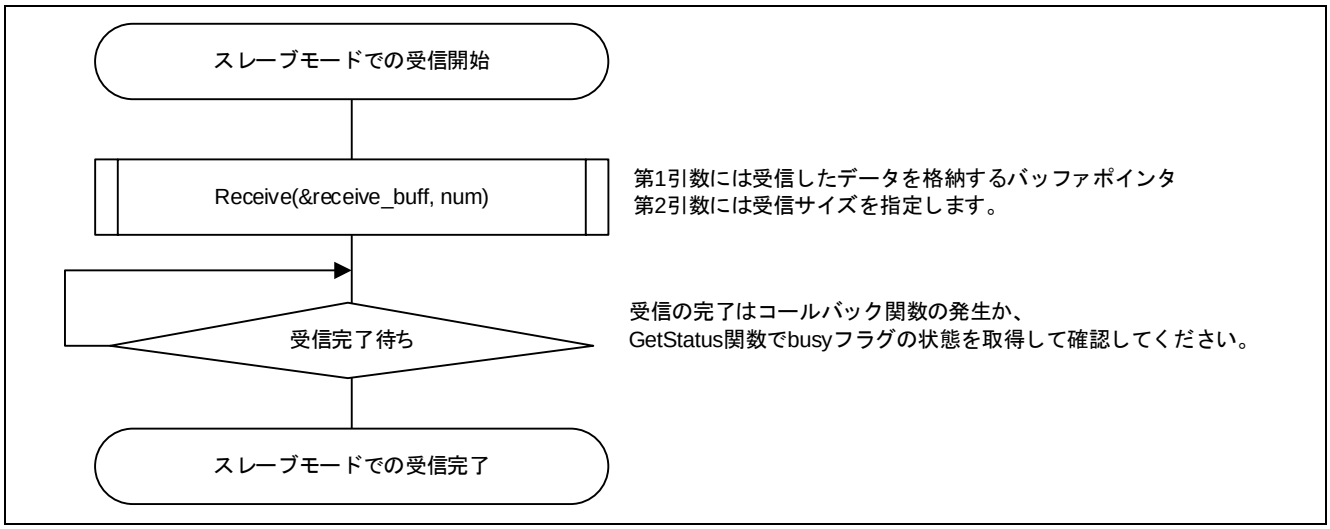


図 3-19 スレーブモードでの受信手順

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_SPI_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、送受信エラーが発生した場合、ARM_SPI_EVENT_DATA_LOST 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。

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、SPI 通信では受信のみの動作ができないため、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_SPI_SET_DEFAULT_TX_VALUE コマンドにて変更できます。

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。

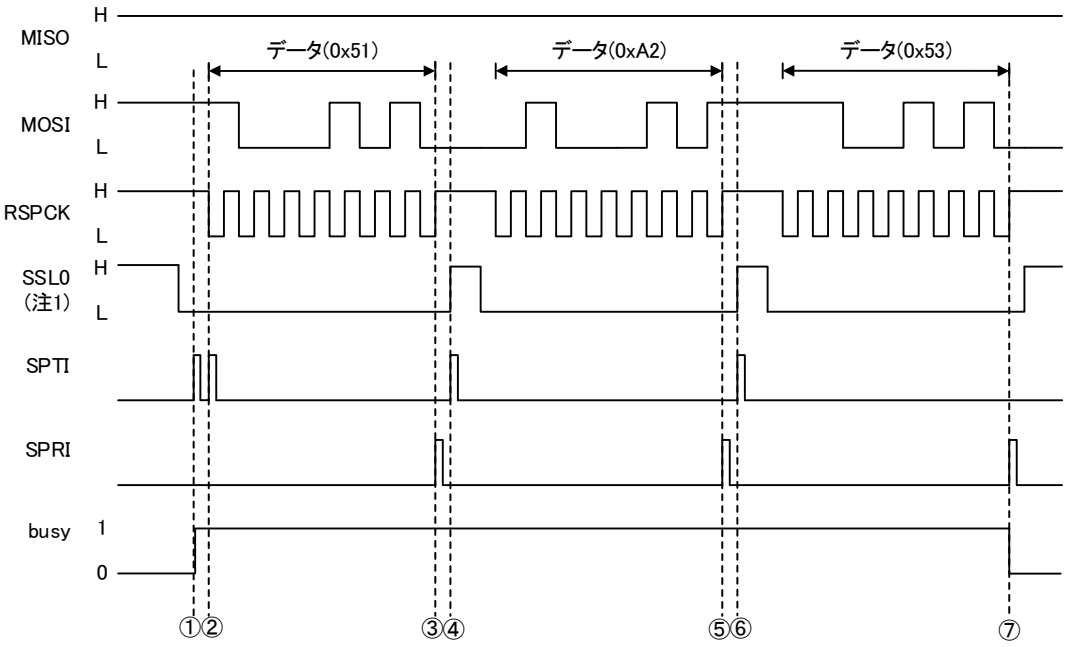


図 3-20 割り込み制御による受信動作（3 バイト受信、ダミーデータ 0xFF）

① Receive 関数を実行すると、busy フラグが”1”(通信状態)になります。また、SPTI 割り込みが発生し、ダミーデータをデータレジスタ(SPDR)に書き込みます。

② RSPCK 端子にクロックが入力されると 1 バイト目のダミーデータが MISO 端子から出力を開始。2 回目の SPTI 割り込みにて 2 バイト目のダミーデータを SPDR レジスタに書き込みます。

③ MOSI 端子からデータを受信すると、SPRI 割り込みが発生し、データレジスタ(SPDR)の値を指定されたバッファに読み出します。

④ 3 回目の SPTI 割り込みにて最終送信データを SPDR に書き込みます。

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

⑥ ④で書き込んだ最終データの出力による SPTI 割り込みで、SPTI 割り込みを禁止にします。

⑦ 最終データ読み出し時の SPRI 割り込みで、SPRI 割り込みを禁止にします。SPRI 割り込みでは、SPI 制御で使用するすべての割り込みを禁止にします。また、busy フラグが”0”（通信待ち状態）になります。コールバック関数が登録されている場合、ARM_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注 1) (注 2)

注1. 送受信エラーが発生した場合、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

注2. CPHA = 0 設定時、通信を再開する場合は RSPCK の半サイクルをソフトウェアで待ってから実行してください。詳細は「5.6 スレーブモードかつ CPHA0 での通信再開について」参照。

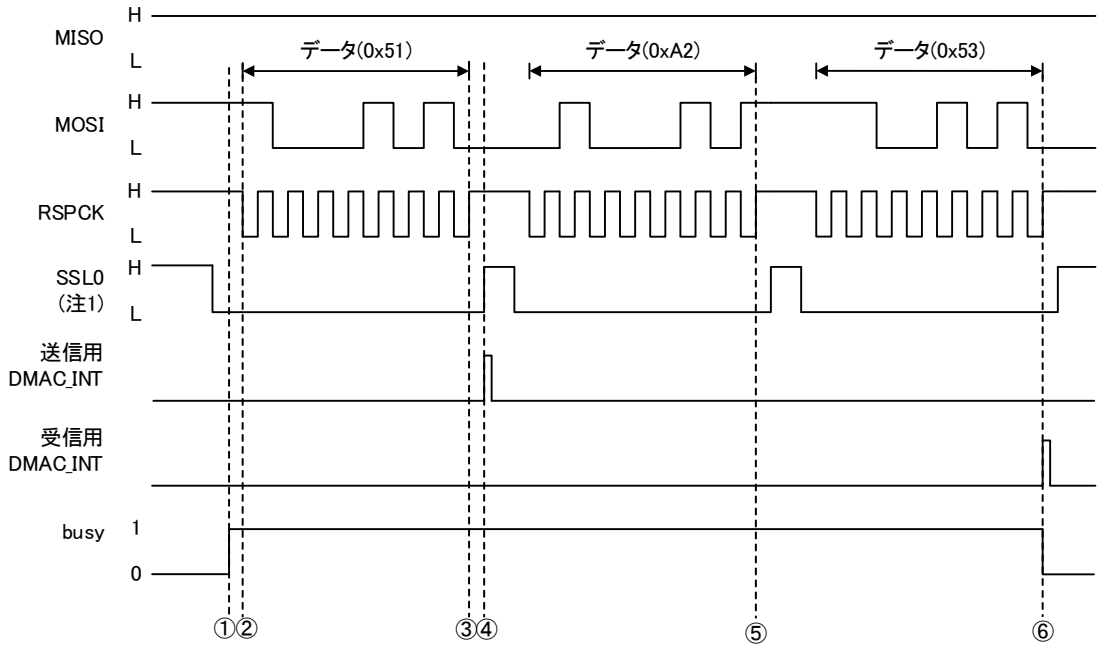


図 3-21 DMAC 制御による受信動作（3 バイト受信、ダミーデータ 0xFF）

① Receive 関数を実行すると、busy フラグを”1”（通信ビジー）に設定、DMAC の転送要因に SPTI 割り込み、SPRI 割り込みを設定します。また、DMA 転送にてダミーデータがデータレジスタ(SPDR)に転送されます。

② RSPCK 端子にクロックが入力されると、ダミーデータが MISO 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(SPDR)に転送されます。

③ MOSI 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(SPDR)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、送信側の DMAC 転送完了割り込みを禁止に設定します。

⑤ 1 バイト受信完了ごとに DMA 転送にて、SPDR 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 busy フラグを”0”（通信待ち状態）にし、SPI 制御で使用するすべての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_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注 1)(注 2)

注1. 受信エラーが発生した場合、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

注2. CPHA = 0 設定時、通信を再開する場合は RSPCK の半サイクルをソフトウェアで待ってから実行してください。詳細は「5.6 スレーブモードかつ CPHA0 での通信再開について」参照。

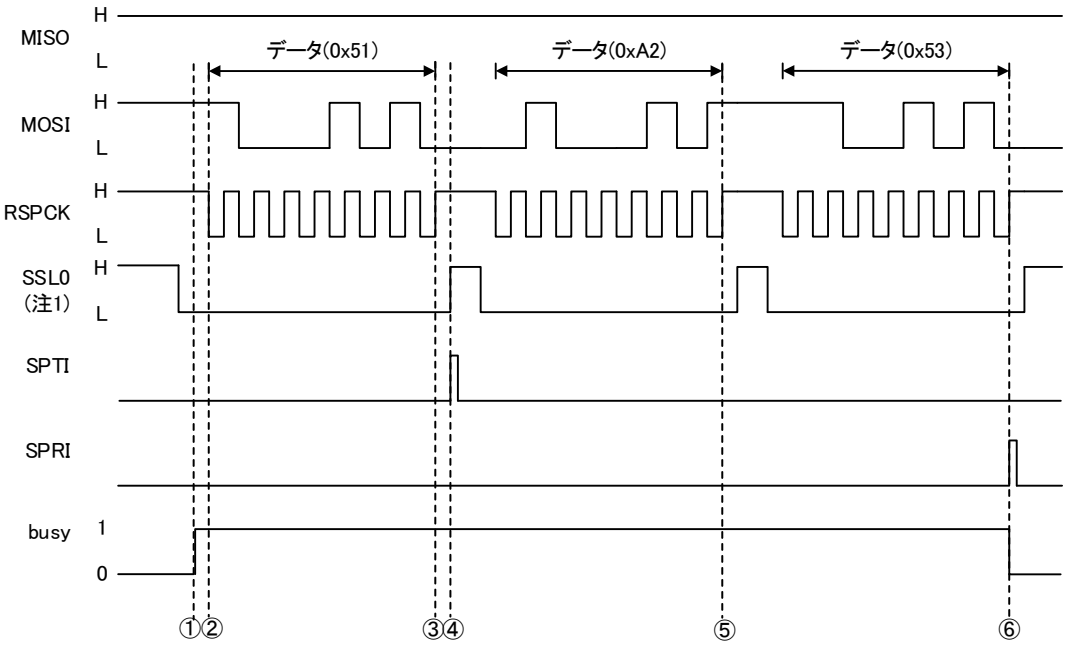


図 3-22 DTC 制御による受信動作（3 バイト受信、ダミーデータ 0xFF）

① Receive 関数を実行すると、busy フラグを”1”（通信ビジー）に設定、DTC の転送要因に SPTI 割り込み、SPRI 割り込みを設定します。また、DMA 転送にてダミーデータがデータレジスタ(SPDR)に転送されます。

② RSPCK 端子にクロックが入力されると、ダミーデータが MISO 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(SPDR)に転送されます。

③ MOSI 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(SPDR)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、SPTI 割り込みが発生します。割り込み処理にて、SPTI 割り込みを禁止に設定します。

⑤ 1 バイト受信完了ごとに DMA 転送にて、SPDR 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、SPRI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、SPI 制御で使用するすべての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_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注 1)(注 2)

注1. 受信エラーが発生した場合、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

注2. CPHA = 0 設定時、通信を再開する場合は RSPCK の半サイクルをソフトウェアで待ってから実行してください。詳細は「5.6 スレーブモードかつ CPHA0 での通信再開について」参照。

3.2.4 スレーブモードでの送受信処理

スレーブモードで送受信を行う手順を図 3-23 に示します。

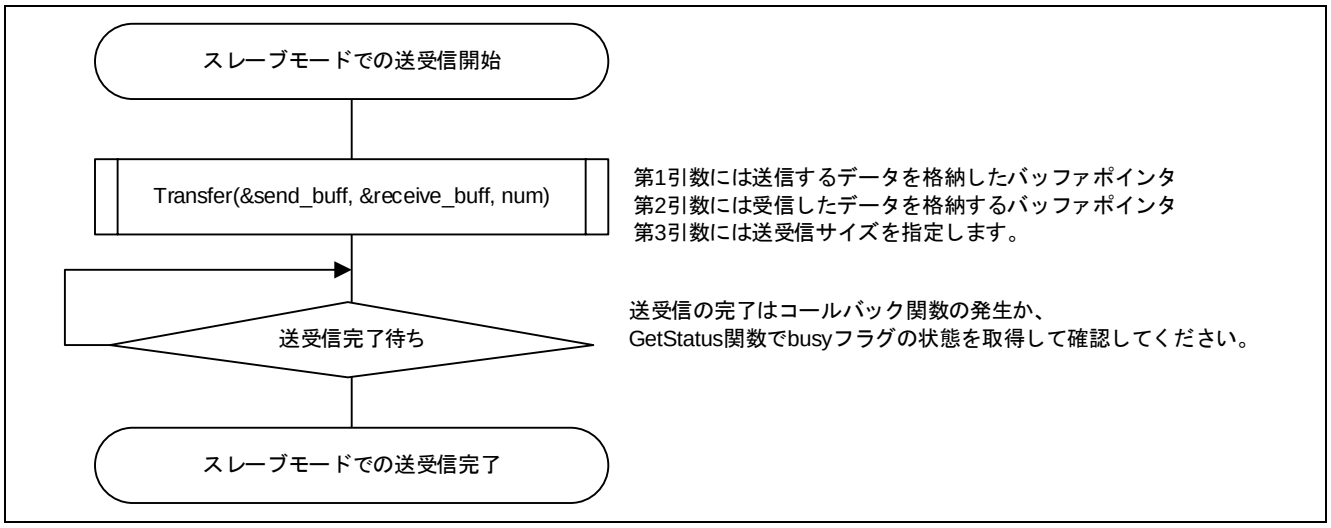


図 3-23 スレーブモードでの送受信手順

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_SPI_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、ARM_SPI_EVENT_DATA_LOST 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。

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 3-24 に通信制御が割り込みの場合の動作を、図 3-25 に通信制御が DMAC の場合の動作を、図 3-26 に通信制御が DTC の場合の動作を示します。

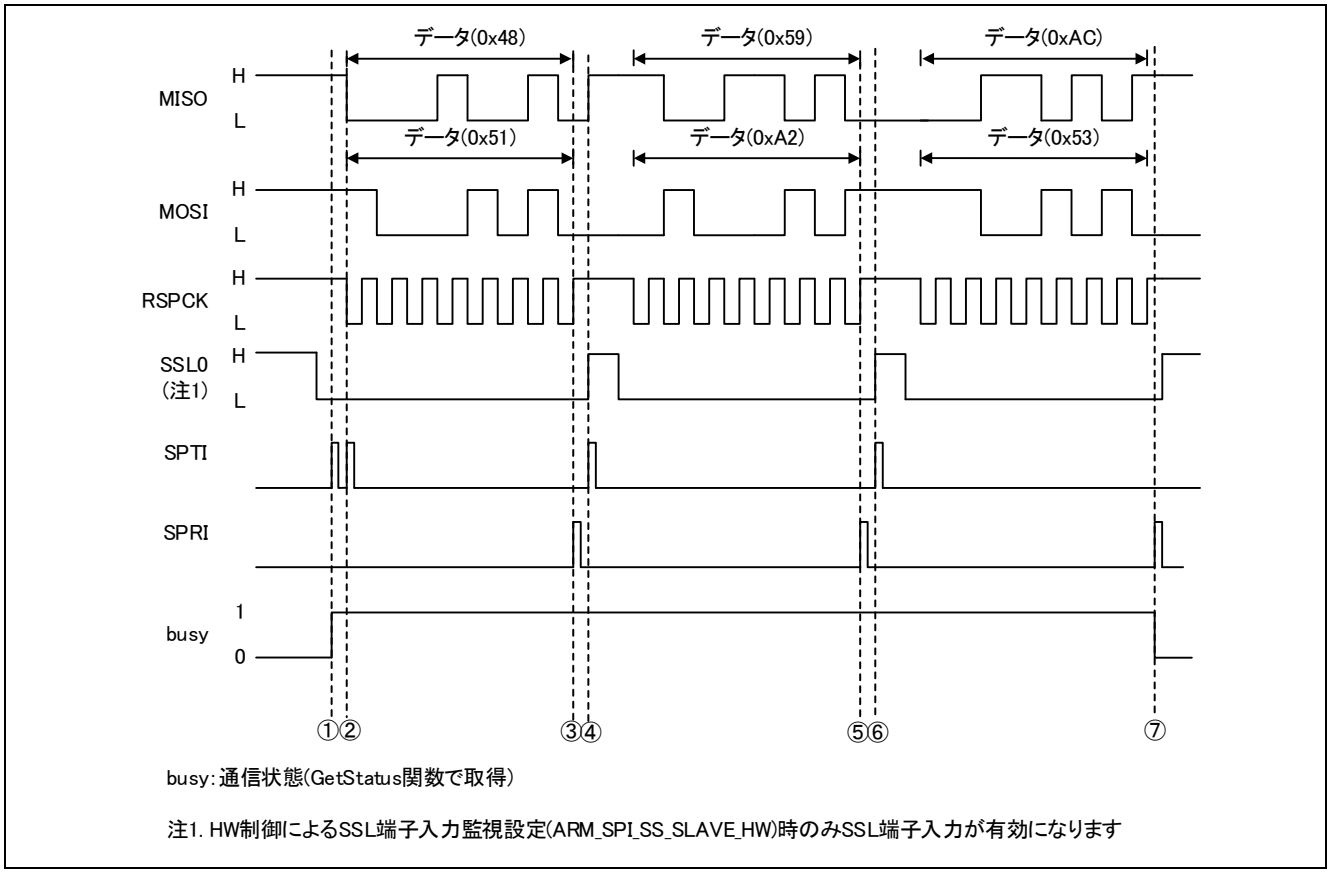


図 3-24 割り込み制御による送受信動作（3 バイト送受信）

① Transfer 関数を実行すると、busy フラグが”1”(通信状態)になります。また、SPTI 割り込みが発生し、1 つ目の送信データをデータレジスタ(SPDR)に書き込みます。

② RSPCK 端子にクロックが入力されると 1 バイト目の送信データが MISO 端子から出力を開始。2 回目の SPTI 割り込みにて 2 バイト目の送信データを SPDR レジスタに書き込みます。

③ MOSI 端子からデータを受信するごとに SPRI 割り込みが発生し、データレジスタ(SPDR)の値を指定されたバッファに読み出します。

④ 3 回目の SPTI 割り込みにて最終送信データを SPDR に書き込みます。

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

⑥ ④で書き込んだ最終データの出力による SPTI 割り込みで、SPTI 割り込みを禁止にします。

⑦ 最終データ読み出し時の SPRI 割り込みで、SPI 制御で使用するすべての割り込みを禁止にします。また、busy フラグが”0”（通信待ち状態）になります。コールバック関数が登録されている場合、ARM_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注 1) (注 2)

注1. 送受信エラーが発生した場合、SPEI 割り込みが発生します。割り込み処理にて busy フラグを” 0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

注2. CPHA = 0 設定時、通信を再開する場合は RSPCK の半サイクルをソフトウェアで待ってから実行してください。詳細は「5.6 スレーブモードかつ CPHA0 での通信再開について」参照。

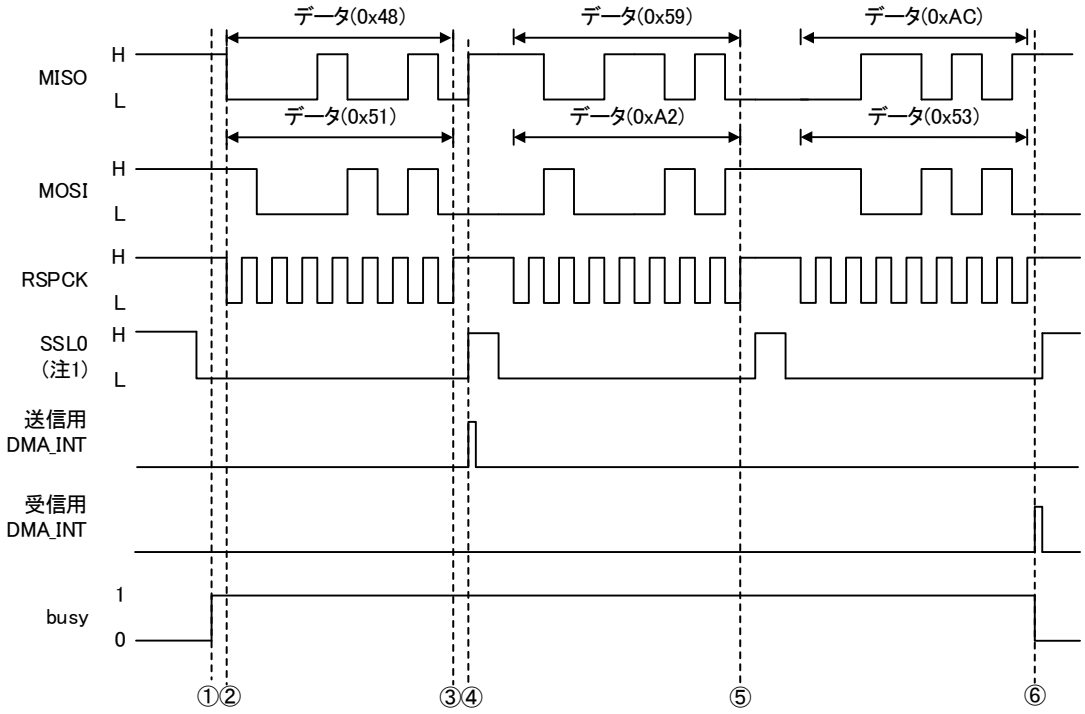


図 3-25 DMAC 制御による送受信動作（3 バイト受信）

① Transfer 関数を実行すると、busy フラグを”1”（通信ビジー）に設定、DMAC の転送要因に SPTI 割り込み、SPRI 割り込みを設定します。また、DMA 転送にて送信データがデータレジスタ(SPDR)に転送されます。

② RSPCK 端子にクロックが入力されると、送信データが MISO 端子から出力を開始、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(SPDR)に転送されます。

③ MOSI 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(SPDR)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時に送信側の DMAC 転送完了割り込みが発生します。割り込み処理にて、送信側の DMAC 転送完了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、SPDR 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 busy フラグを”0”（通信待ち状態）にし、SPI 制御で使用するすべての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_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注 1)(注 2)

注1. 送受信エラーが発生した場合、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

注2. CPHA = 0 設定時、通信を再開する場合は RSPCK の半サイクルをソフトウェアで待ってから実行してください。詳細は「5.6 スレーブモードかつ CPHA0 での通信再開について」参照。

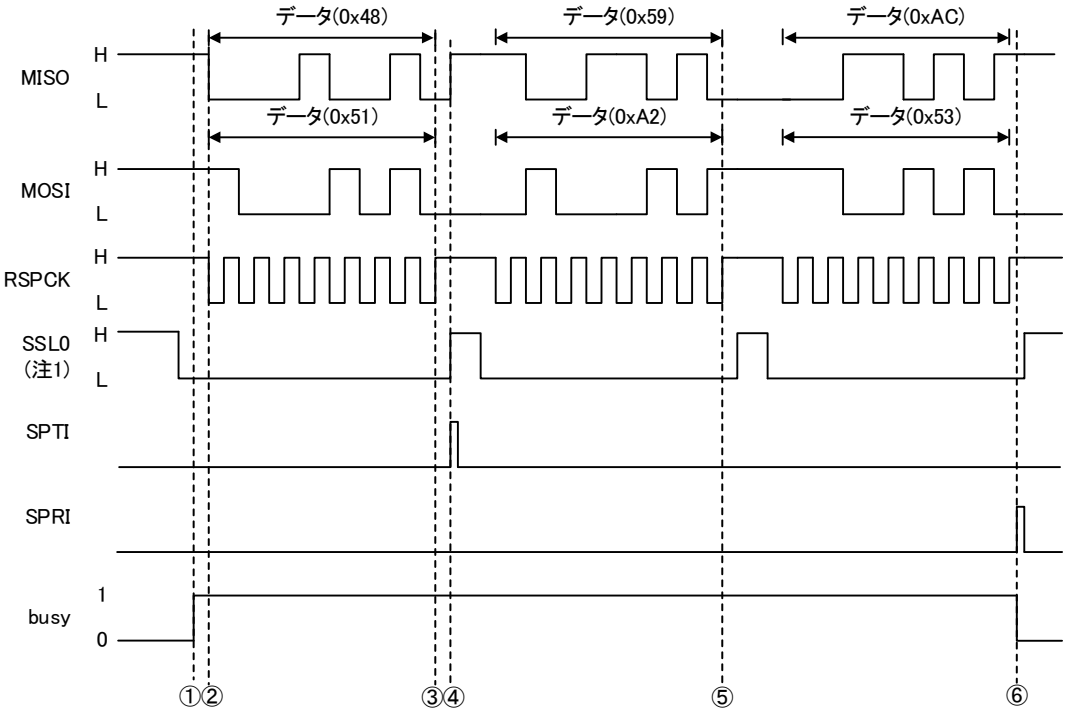


図 3-26 DTC 制御による送受信動作（3 バイト受信）

① Transfer 関数を実行すると、busy フラグを”1”（通信ビジー）に設定、DTC の転送要因に SPTI 割り込み、SPRI 割り込みを設定します。また、DMA 転送にて送信データがデータレジスタ(SPDR)に転送されます。

② RSPCK 端子にクロックが入力されると、送信データが MISO 端子から出力を開始、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(SPDR)に転送されます。

③ MOSI 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(SPDR)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、SPTI 転送完了割り込みが発生します。割り込み処理にて SPTI 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、SPDR 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、SPRI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、SPI 制御で使用するすべての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_SPI_EVENT_TRANSFER_COMPLETE を引数にコールバック関数を実行します。(注 1)(注 2)

注1. 送受信エラーが発生した場合、SPEI 割り込みが発生します。割り込み処理にて busy フラグを”0”（通信待ち状態）にし、エラー状態をクリア、SPI 制御で使用するすべての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_SPI_EVENT_DATA_LOST を引数にコールバック関数を実行します。

注2. CPHA = 0 設定時、通信を再開する場合は RSPCK の半サイクルをソフトウェアで待ってから実行してください。詳細は「5.6 スレーブモードかつ CPHA0 での通信再開について」参照。

3.3 コンフィグレーション

SPI 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_spi_cfg.h ファイルに用意します。

3.3.1 送信制御設定

送信制御方法を設定します。

名称：SPIn_TRANSMIT_CONTROL (n = 0、1)

表 3-1 SPIn_TRANSMIT_CONTROL の設定

設定値	内容
SPI_USED_INTERRUPT（初期値）	送信制御に割り込みを使用
SPI_USED_DMAC0	送信制御に DMAC0 を使用
SPI_USED_DMAC1	送信制御に DMAC1 を使用
SPI_USED_DMAC2	送信制御に DMAC2 を使用
SPI_USED_DMAC3	送信制御に DMAC3 を使用
SPI_USED_DTC	送信制御に DTC を使用

3.3.2 受信制御設定

受信制御方法を設定します。

名称：SPIn_RECEIVE_CONTROL (n = 0、1)

表 3-2 SPIn_RECEIVE_CONTROL の設定

設定値	内容
SPI_USED_INTERRUPT（初期値）	受信制御に割り込みを使用
SPI_USED_DMAC0	受信制御に DMAC0 を使用
SPI_USED_DMAC1	受信制御に DMAC1 を使用
SPI_USED_DMAC2	受信制御に DMAC2 を使用
SPI_USED_DMAC3	受信制御に DMAC3 を使用
SPI_USED_DTC	受信制御に DTC を使用

3.3.3 SPTI 割り込み優先レベル

SPTIn 割り込みの優先レベルを設定します。（n = 0、1）

名称：SPIn_SPTI_PRIORITY

表 3-3 SPIn_SPTI_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

3.3.4 SPRI 割り込み優先レベル

SPRIn 割り込みの優先レベルを設定します。（n = 0、1）

名称：SPIn_SPRI_PRIORITY

表 3-4 SPIn_SPRI_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

3.3.5 SPII 割り込み優先レベル

SPIIn 割り込みの優先レベルを設定します。（n = 0、1）

名称：SPIn_SPII_PRIORITY

表 3-5 SPIn_SPII_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

3.3.6 SPEI 割り込み優先レベル

SPEIn 割り込みの優先レベルを設定します。（n = 0、1）

名称：SPIn_SPEI_PRIORITY

表 3-6 SPIn_SPEI_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

3.3.7 SPTEND 割り込み優先レベル

SPTENDn 割り込みの優先レベルを設定します。（n = 0、1）

名称：SPIn_SPTEND_PRIORITY

表 3-7 SPIn_SPTEND_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

3.3.8 ソフトウェア制御による SSL 端子定義

ソフトウェア制御にて使用する SSL 端子を定義します。

名称：SPIn_SS_PORT(注)、SPIn_SS_PIN（n = 0、1）

表 3-8 SPIn_SS_PORT、SPIn_SS_PIN の設定

名称	初期値	内容
SPIn_SS_PORT(注)	(PORT0->PODR)	SSL 端子に PORT0 を選択
SPIn_SS_PIN	(0)	SSL 端子に PORTi00 を選択 (i は SPIn_SS_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 SSL 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。

3.3.9 関数の RAM 配置

SPI 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：SPI_CFG_SECTION_xxx

xxx には関数名をすべて大文字で記載

例）　ARM_SPI_INITIALIZE 関数　→　SPI_CFG_SECTION_ARM_SPI_INITIALIZE

表 3-9 SPI_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 3-10 各関数の RAM 配置初期状態

番号	関数名	RAM 配置
1	ARM_SPI_GetVersion	
2	ARM_SPI_GetCapabilities	
3	ARM_SPI_Initialize	
4	ARM_SPI_Uninitialize	
5	ARM_SPI_PowerControl	
6	ARM_SPI_Send	
7	ARM_SPI_Receive	
8	ARM_SPI_Transfer	
9	ARM_SPI_GetDataCount	
10	ARM_SPI_Control	
11	ARM_SPI_GetStatus	
12	spin_spti_interrupt（n = 0、1）（SPTI 割り込み処理）	✔
13	spin_spri_interrupt（n = 0、1）（SPRI 割り込み処理）	✔
14	spin_spii_interrupt（n = 0、1）（SPII 割り込み処理）	✔
15	spin_spei_interrupt（n = 0、1）（SPEI 割り込み処理）	✔
16	spin_sptend_interrupt（n = 0、1）（SPTEND 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

SPI 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_SPI_GetVersion 関数

表 4-1 ARM_SPI_GetVersion 関数仕様

書式	ARM_DRIVER_VERSION ARM_SPI_GetVersion(void)
仕様説明	SPI ドライバのバージョンを取得します
引数	なし
戻り値	SPI 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]
	// SPI driver instance (SPI0)
	extern ARM_DRIVER_SPI Driver_SPI0;
	ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
	
	main()
	{
	ARM_DRIVER_VERSION version;
	version = spi0Drv->GetVersion();
	
	}

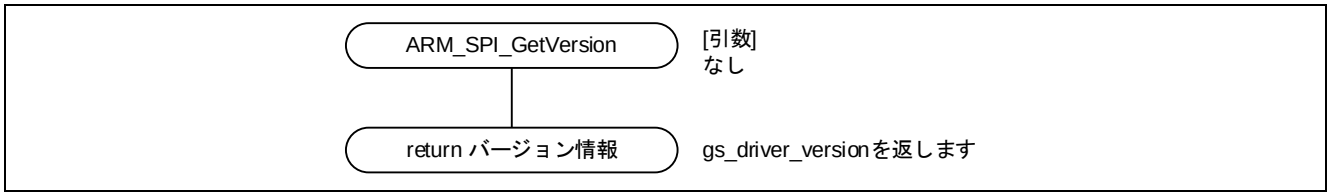


図 4-1 ARM_SPI_GetVersion 関数処理フロー

4.1.2 ARM_SPI_GetCapabilities 関数

表 4-2 ARM_SPI_GetCapabilities 関数仕様

書式	ARM_SPI_CAPABILITIES ARM_SPI_GetCapabilities(void)
仕様説明	SPI 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]
	// SPI driver instance (SPI0)
	extern ARM_DRIVER_SPI Driver_SPI0;
	ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
	
	main()
	{
	ARM_SPI_CAPABILITIES cap;
	cap = spi0Drv->GetCapabilities();
	
	}

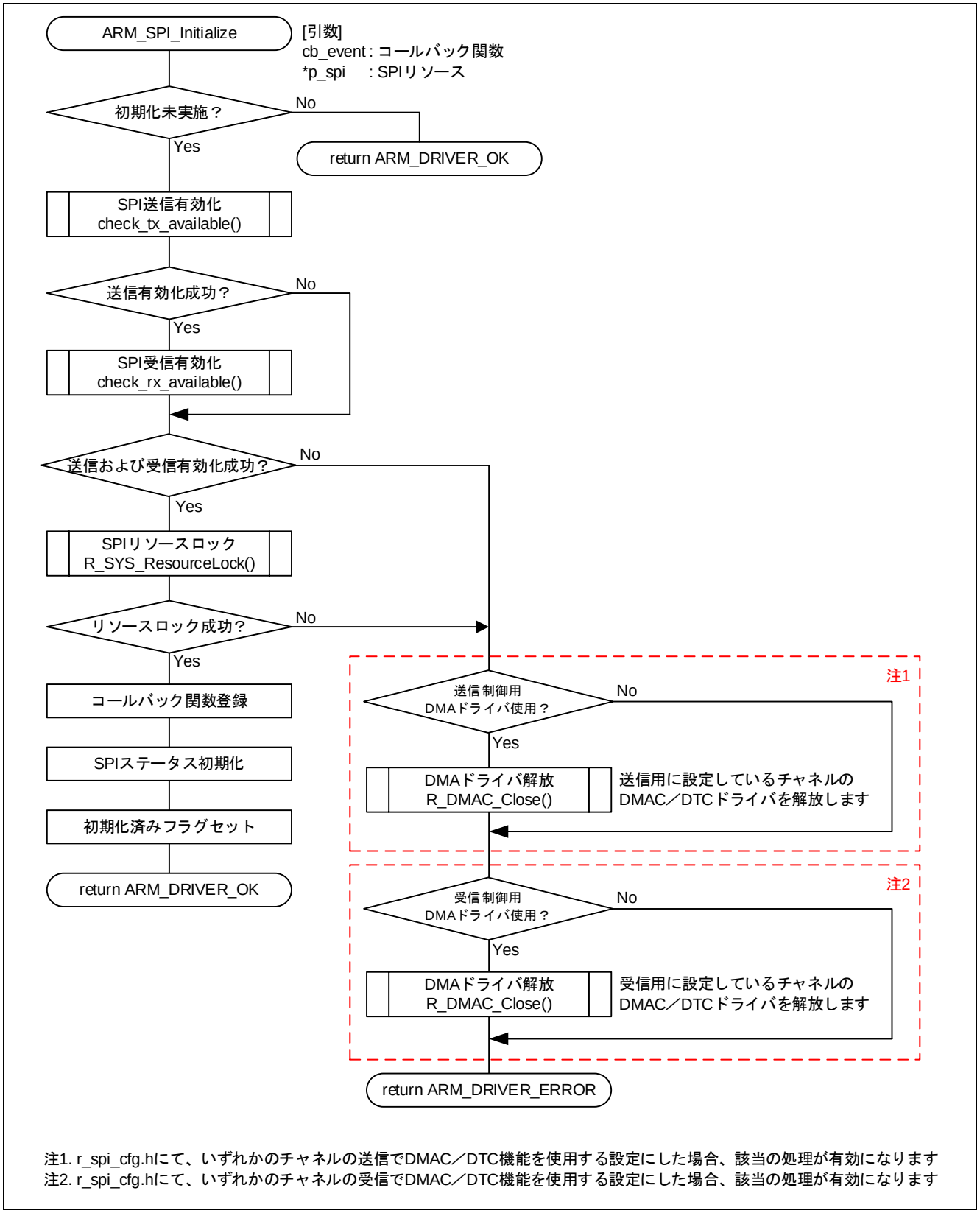
図 4-2 ARM_SPI_GetCapabilities 関数処理フロー

4.1.3 ARM_SPI_Initialize 関数

表 4-3 ARM_SPI_Initialize 関数仕様

書式	int32_t ARM_SPI_Initialize(ARM_SPI_SignalEvent_t cb_event, st_spi_resources_t * const p_spi)
仕様説明	SPI 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_SPI_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_spi_resources_t * const p_spi : SPI のリソース 初期化する SPI のリソースを指定します
戻り値	ARM_DRIVER_OK　　SPI の初期化成功
	ARM_DRIVER_ERROR　　SPI 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 SPI 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 SPIn 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 SPI 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);
	
	// SPI driver instance (SPI0)
	extern ARM_DRIVER_SPI Driver_SPI0;
	ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
	
	main()
	{
	　　spi0Drv->Initialize(callback);
	}



注1. r_spi_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2. r_spi_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_SPI_Initialize 関数処理フロー

4.1.4 ARM_SPI_Uninitialize 関数

表 4-4 ARM_SPI_Uninitialize 関数仕様

書式	int32_t ARM_SPI_Uninitialize(st_spi_resources_t * const p_spi)
仕様説明	SPI ドライバを解放します
引数	st_spi_resources_t * const p_spi: SPI のリソース 解放する SPI のリソースを指定します
戻り値	ARM_DRIVER_OK SPI の解放成功
	ARM_DRIVER_ERROR SPI の解放失敗 パワーオフ状態かつモジュールスタート状態で実行した場合（R_LPM_ModuleStart にてエラーが発生した場合）、SPI の解放失敗となります

備考	インスタンスからのアクセス時は SPI 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]
	// SPI driver instance (SPI0)
	extern ARM_DRIVER_SPI Driver_SPI0;
	ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
	
	main()
	{
	spi0Drv->Uninitialize();
	}

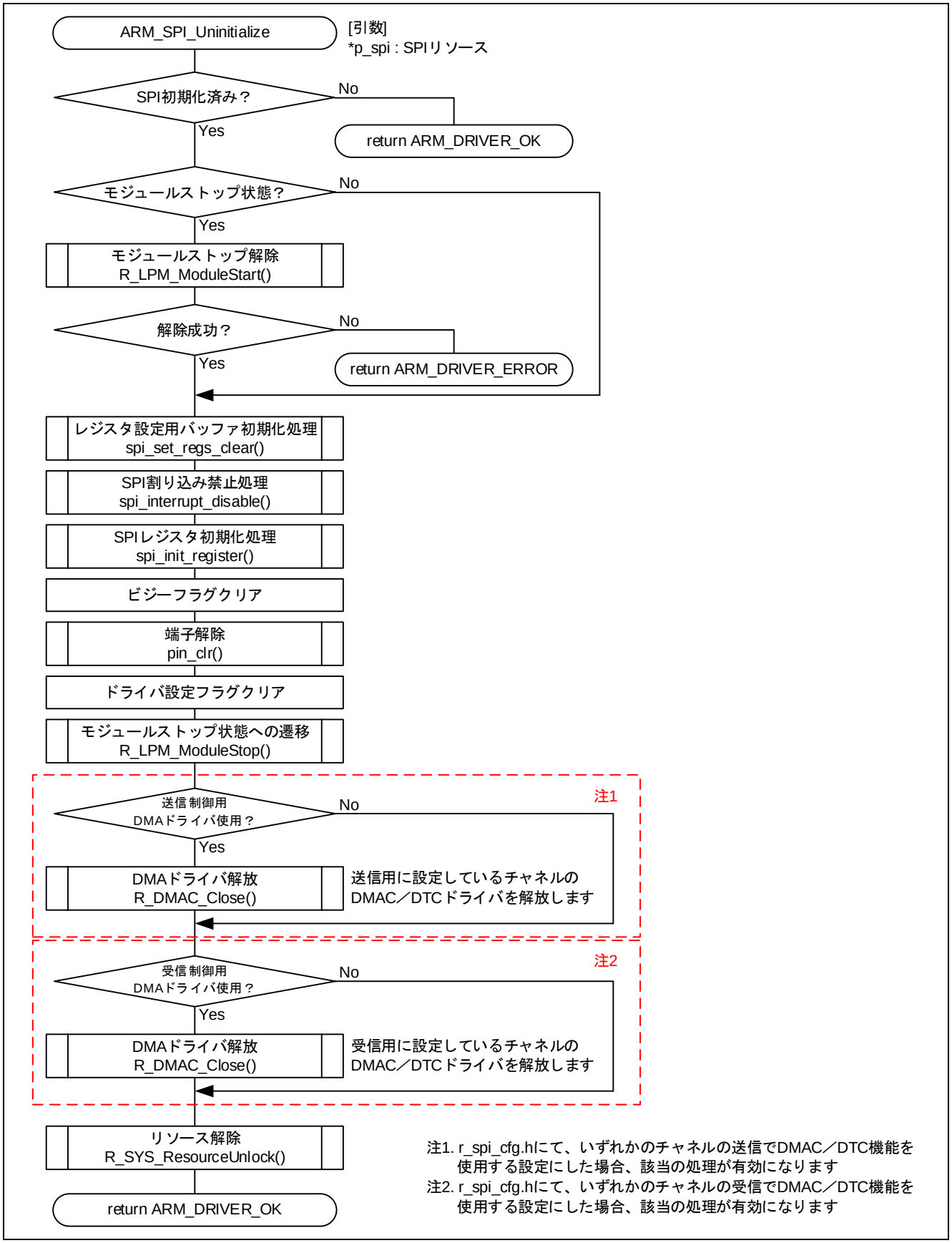


図 4-4 ARM_SPI_Uninitailze 関数処理フロー

4.1.5 ARM_SPI_PowerControl 関数

表 4-5 ARM_SPI_PowerControl 関数仕様

書式	int32_t ARM_SPI_PowerControl(ARM_POWER_STATE state, st_spi_resources_t * const p_spi)
仕様説明	SPI 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_spi_resources_t * const p_spi: SPI のリソース 電源供給する SPI 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 ・SPI 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 SPI 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]
	// SPI driver instance (SPI0)
	extern ARM_DRIVER_SPI Driver_SPI0;
	ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
	
	main()
	{
	spi0Drv-> PowerControl (ARM_POWER_FULL);
	}

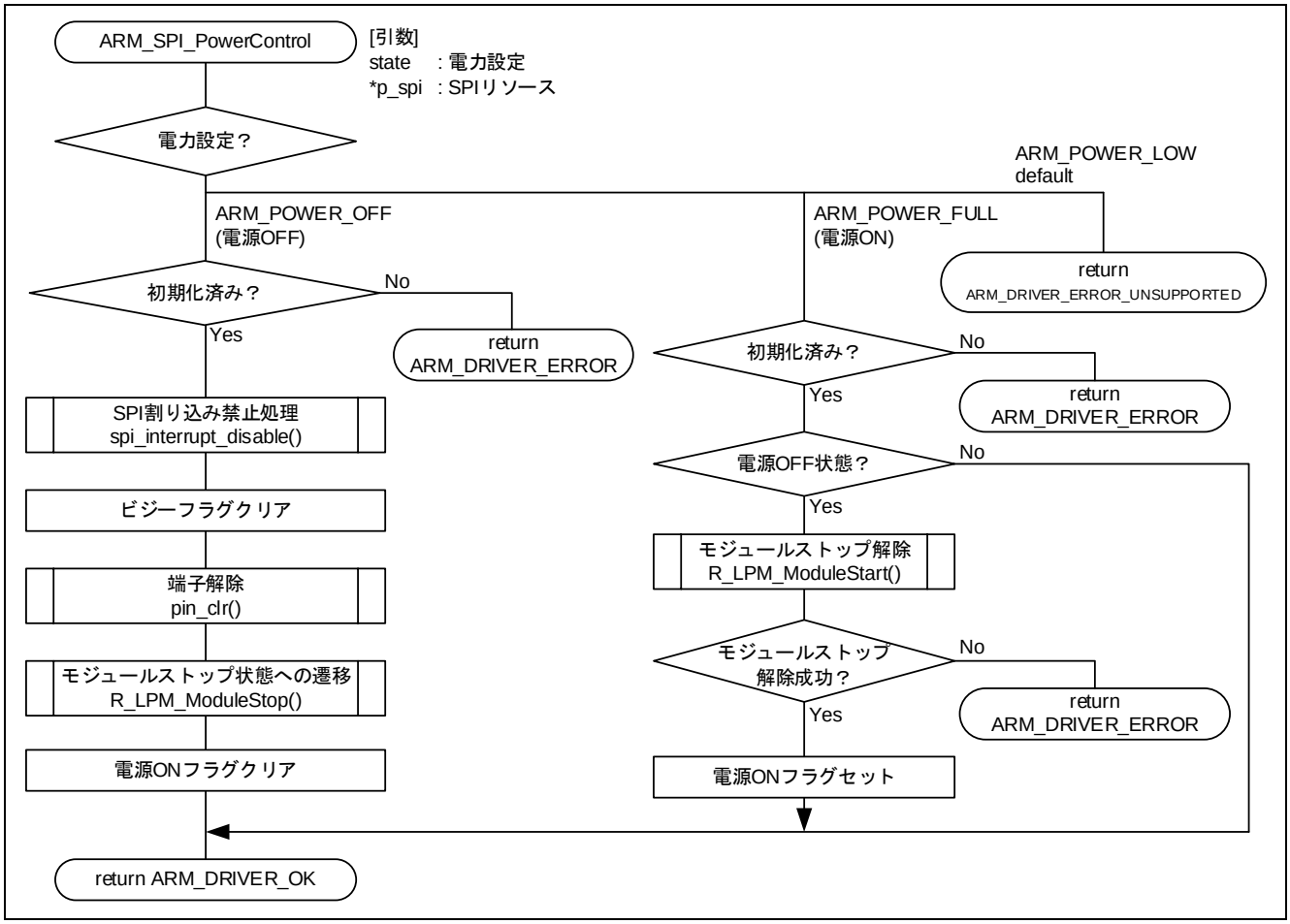


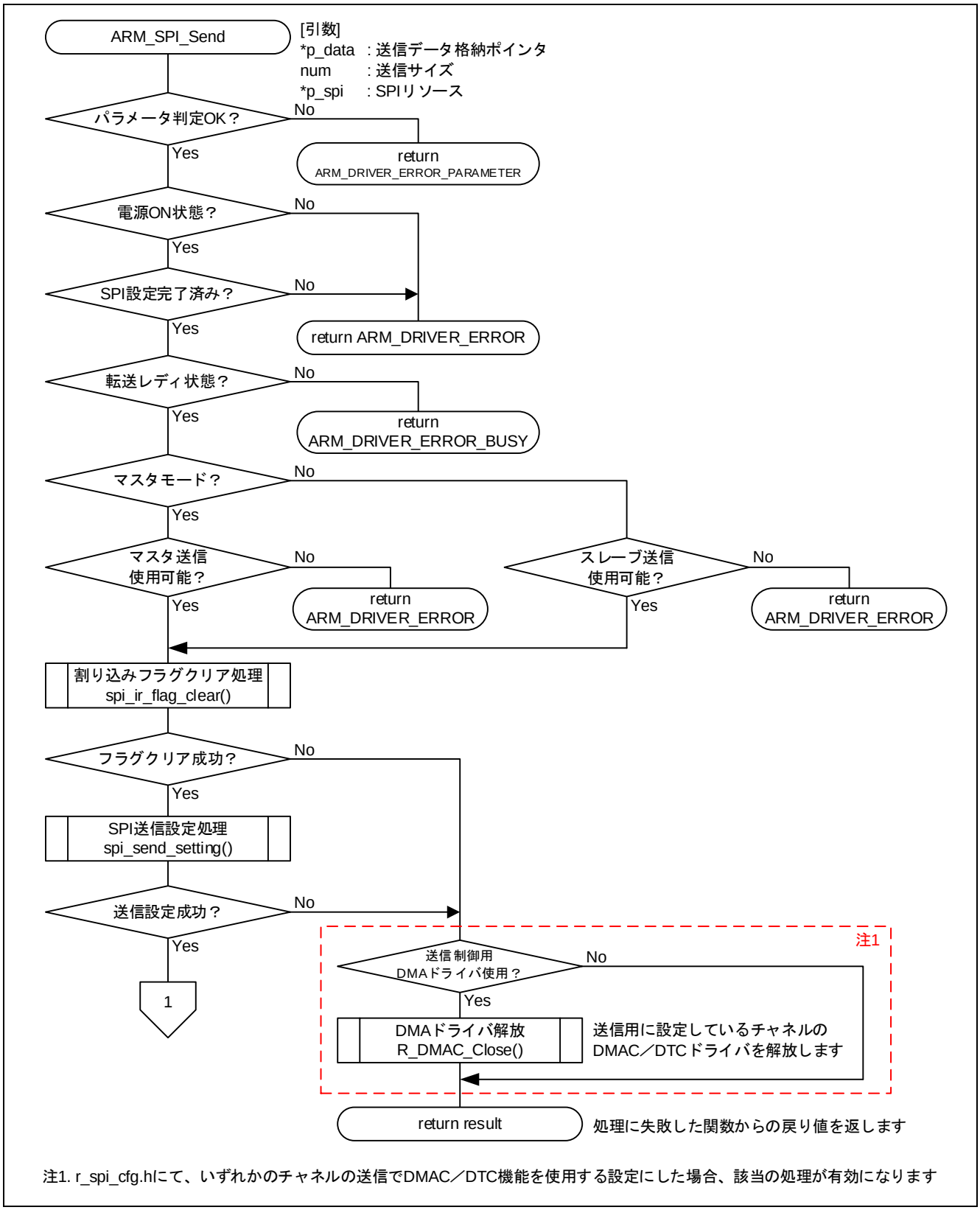
図 4-5 ARM_SPI_PowerControl 関数処理フロー

4.1.6 ARM_SPI_Send 関数

表 4-6 ARM_SPI_Send 関数仕様

書式	int32_t ARM_SPI_Send(void const * const p_data, uint32_t num, st_spi_resources_t * const p_spi)
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_spi_resources_t * const p_spi : SPI のリソース 送信する SPI 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 ・パワーオフ状態で実行した場合 ・未初期化状態で実行した場合 ・マスタモード設定かつマスタ送信動作不可状態で実行した場合 ・スレーブモード設定かつスレーブ送信動作不可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 通信中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
	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 SPI 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]
	// SPI driver instance (SPI0)
	extern ARM_DRIVER_SPI Driver_SPI0;
	ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
	const uint8_t tx_data[2] = {0x51, 0xA2};
	
	main()
	{
	spi0Drv->Send(&tx_data[0], 2);
	}



注1. r_spi_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_SPI_Send 関数処理フロー(1/2)

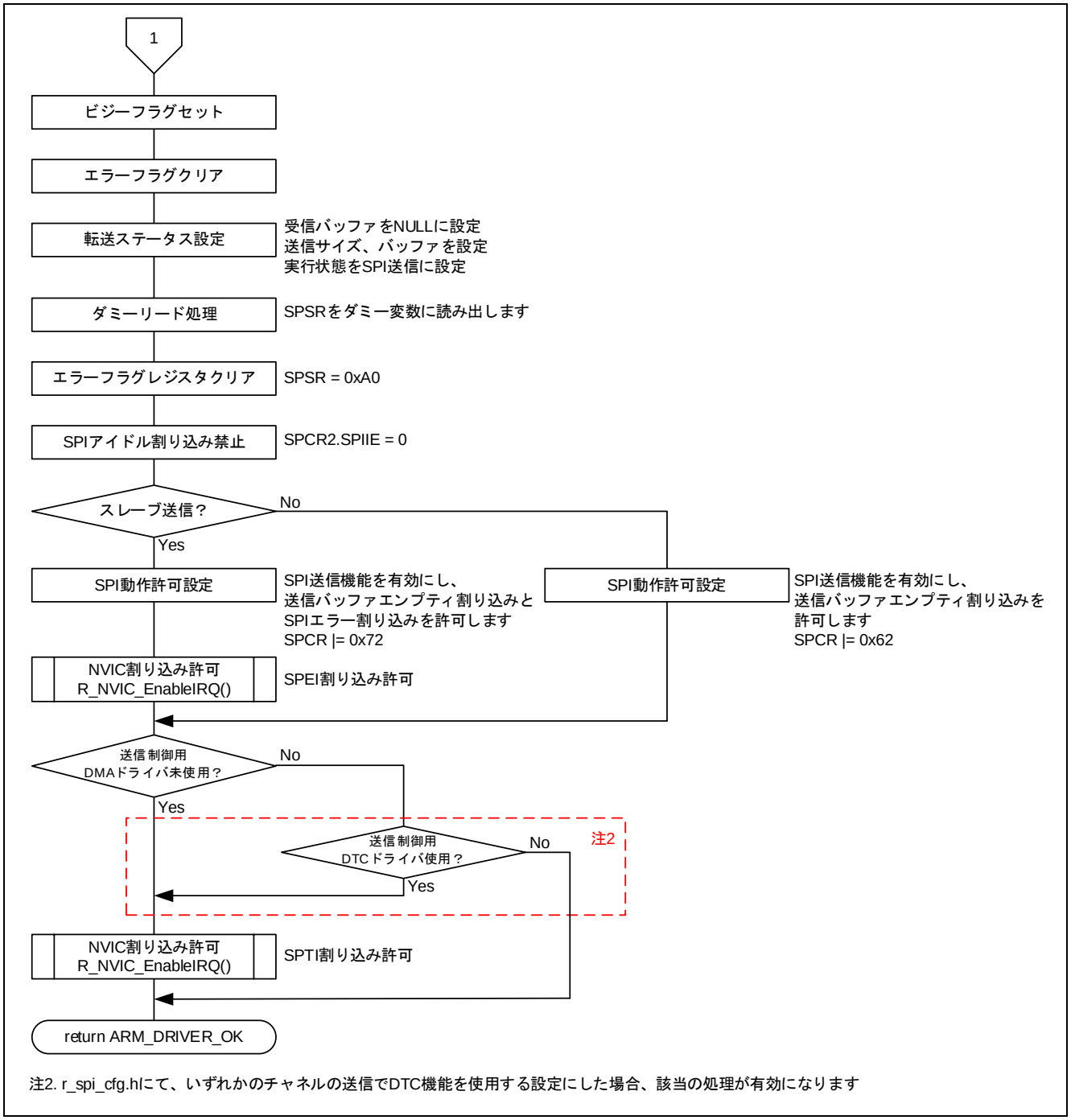


図 4-7 ARM_SPI_Send 関数処理フロー(2/2)

4.1.7 ARM_SPI_Receive 関数

表 4-7 ARM_SPI_Receive 関数仕様

書式	int32_t ARM_SPI_Receive(void const * const p_data, uint32_t num, st_spi_resources_t * const p_spi)
仕様説明	受信を開始します
引数	void const * 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_spi_resources_t * const p_spi : SPI のリソース 受信する SPI 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 ・パワーオフ状態で実行した場合 ・未初期化状態で実行した場合 ・マスタモード設定かつマスタ受信動作不可状態で実行した場合 ・スレーブモード設定かつスレーブ受信動作不可状態で実行した場合 ・送受信処理に 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 通信中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
	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 SPI 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]
	// SPI driver instance (SPI0)
	extern ARM_DRIVER_SPI Driver_SPI0;
	ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
	uint8_t rx_data[2];
	
	main()
	{
	spi0Drv->Receive(&rx_data[0], 2);
	}

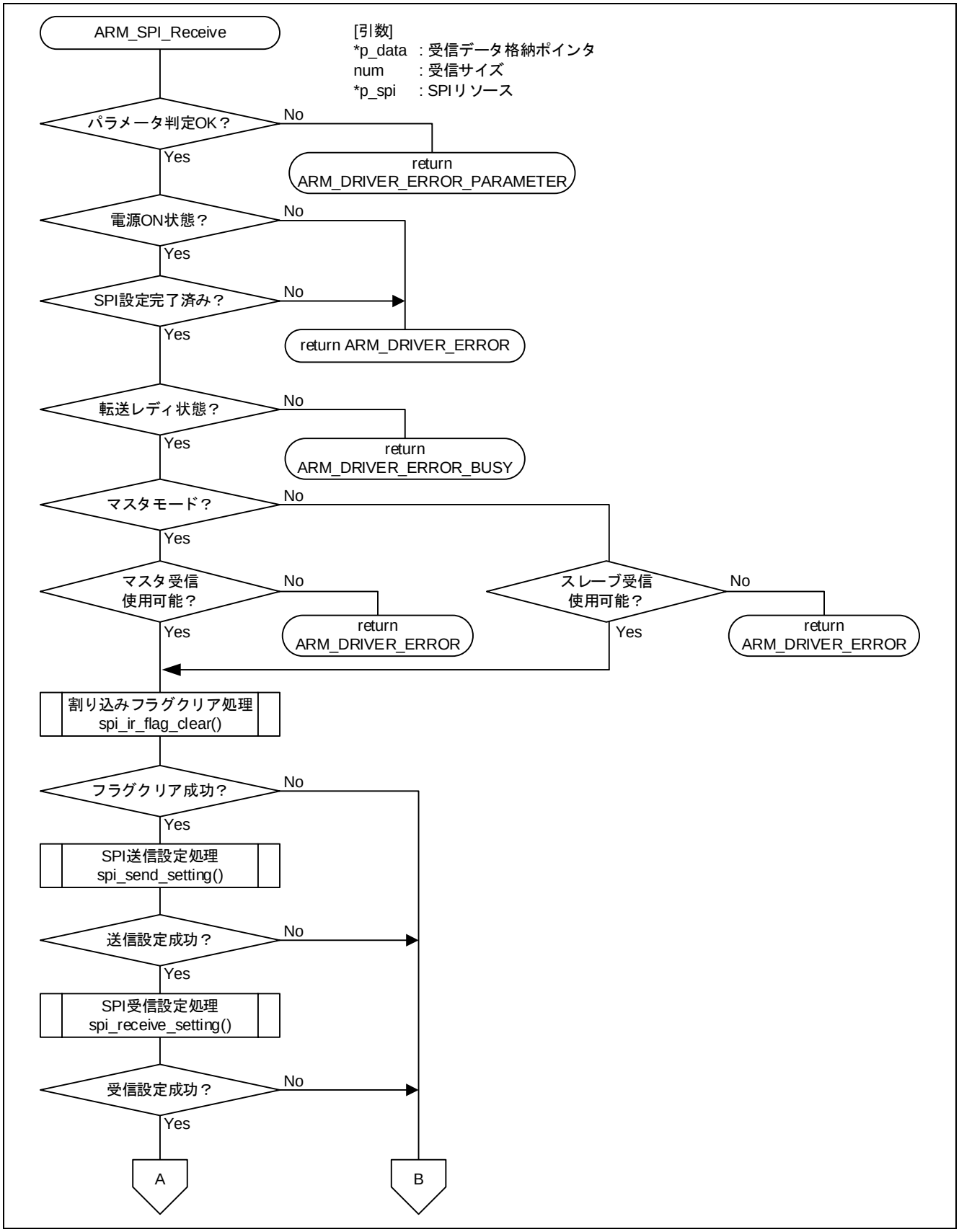
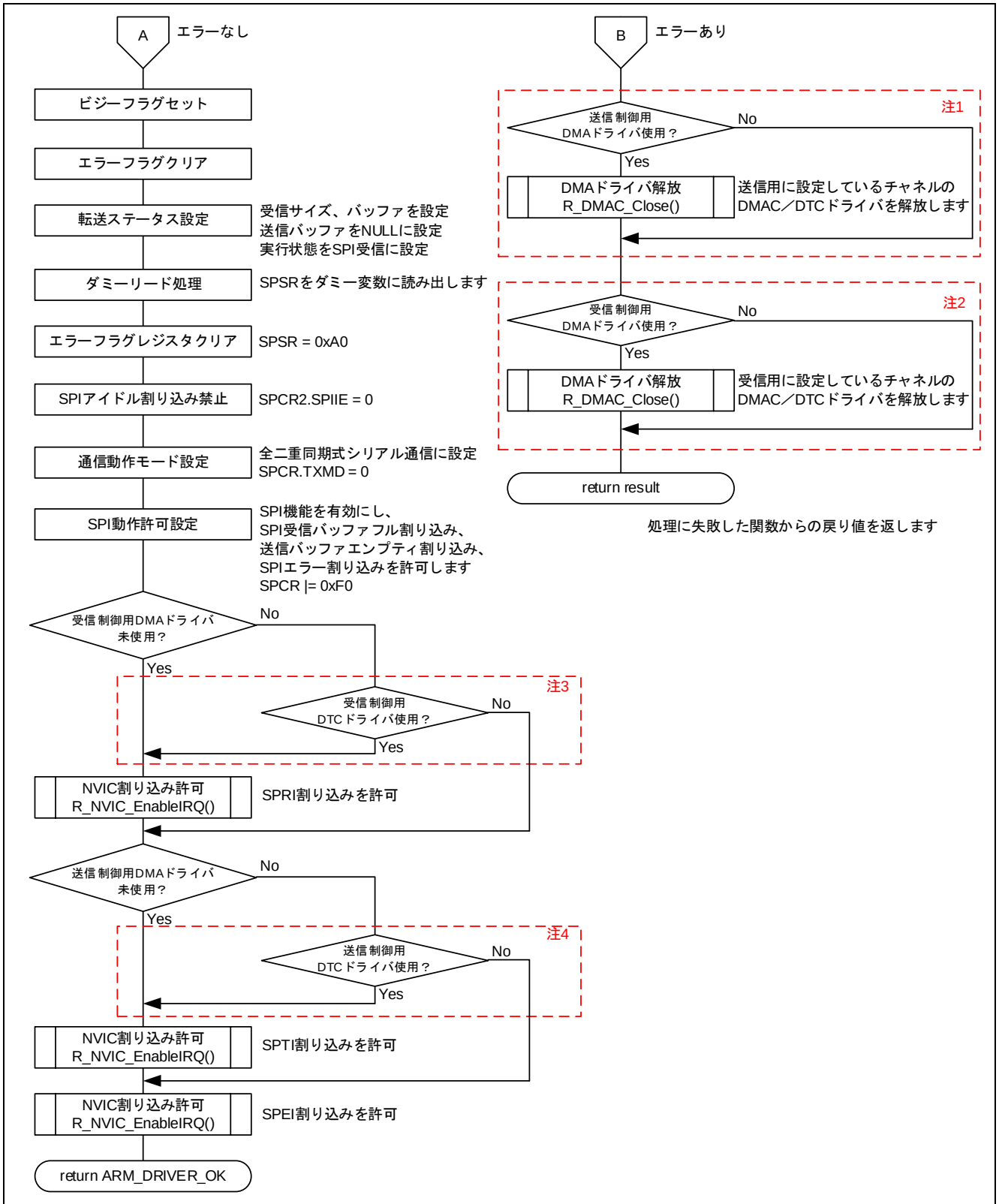


図 4-8 ARM_SPI_Receive 関数処理フロー(1/2)



注1. r_spi_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2. r_spi_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3. r_spi_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_spi_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

図 4-9 ARM_SPI_Receive 関数処理フロー(2/2)

4.1.8 ARM_SPI_Transfer 関数

表 4-8 ARM_SPI_Transfer 関数仕様

書式	int32_t ARM_SPI_Transfer(void const * const p_data_out, void const * const p_data_in, uint32_t num, st_spi_resources_t * const p_spi)
仕様説明	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 * 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_spi_resources_t * const p_spi : SPI のリソース 受信する SPI 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 ・パワーオフ状態で実行した場合 ・未初期化状態で実行した場合 ・マスタモード設定かつマスタ受信動作不可状態で実行した場合 ・スレーブモード設定かつスレーブ受信動作不可状態で実行した場合 ・送受信処理に 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受信失敗 通信中状態を検出するとビジー状態による送受信失敗となります
	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 SPI 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]
	// SPI driver instance (SPI0)
	extern ARM_DRIVER_SPI Driver_SPI0;
	ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
	uint8_t rx_data[2];
	const uint8_t tx_data[2] = {0x51, 0xA2};
	
	main()
	{
	spi0Drv->Transfer (&tx_data[0], &rx_data[0], 2);
	}

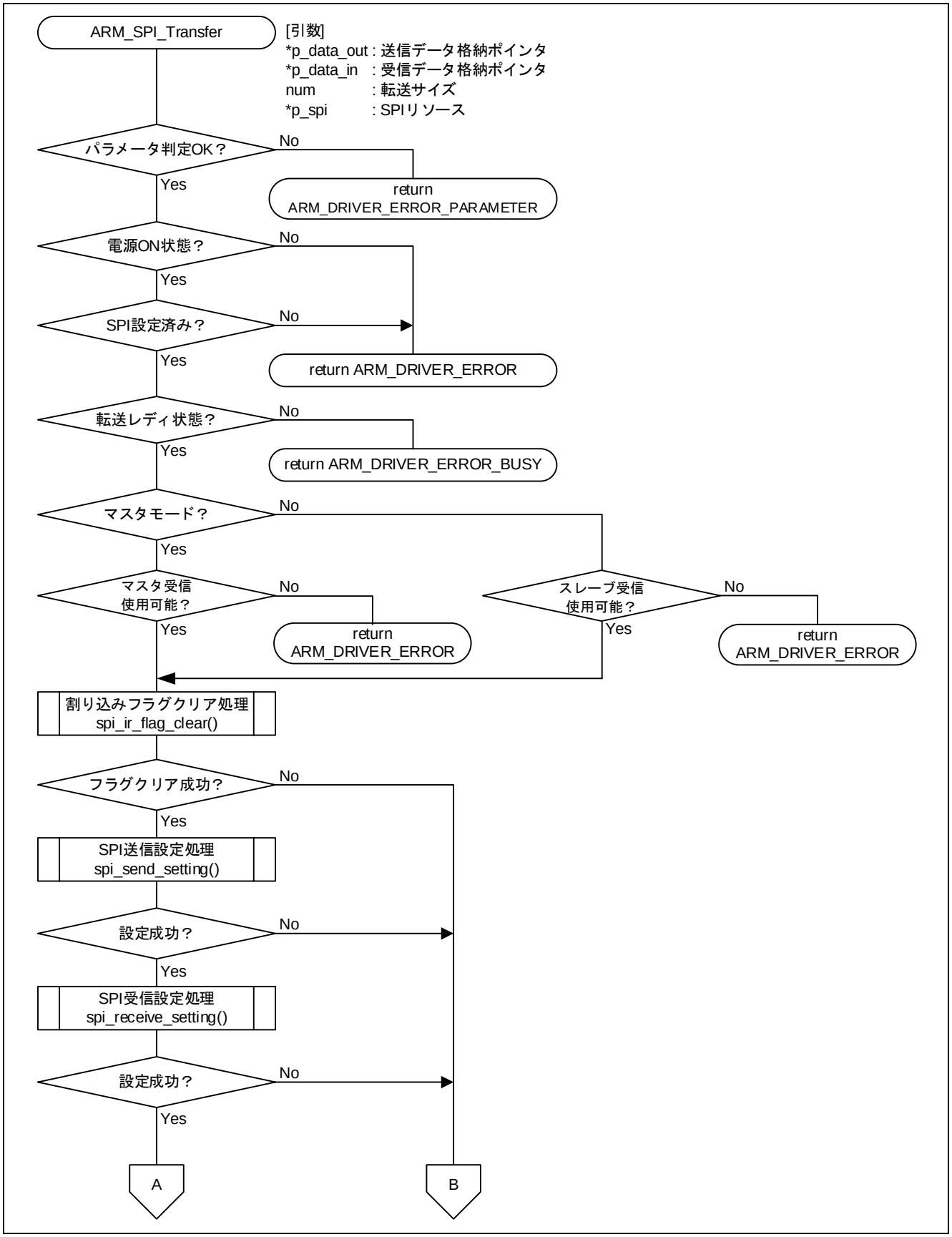
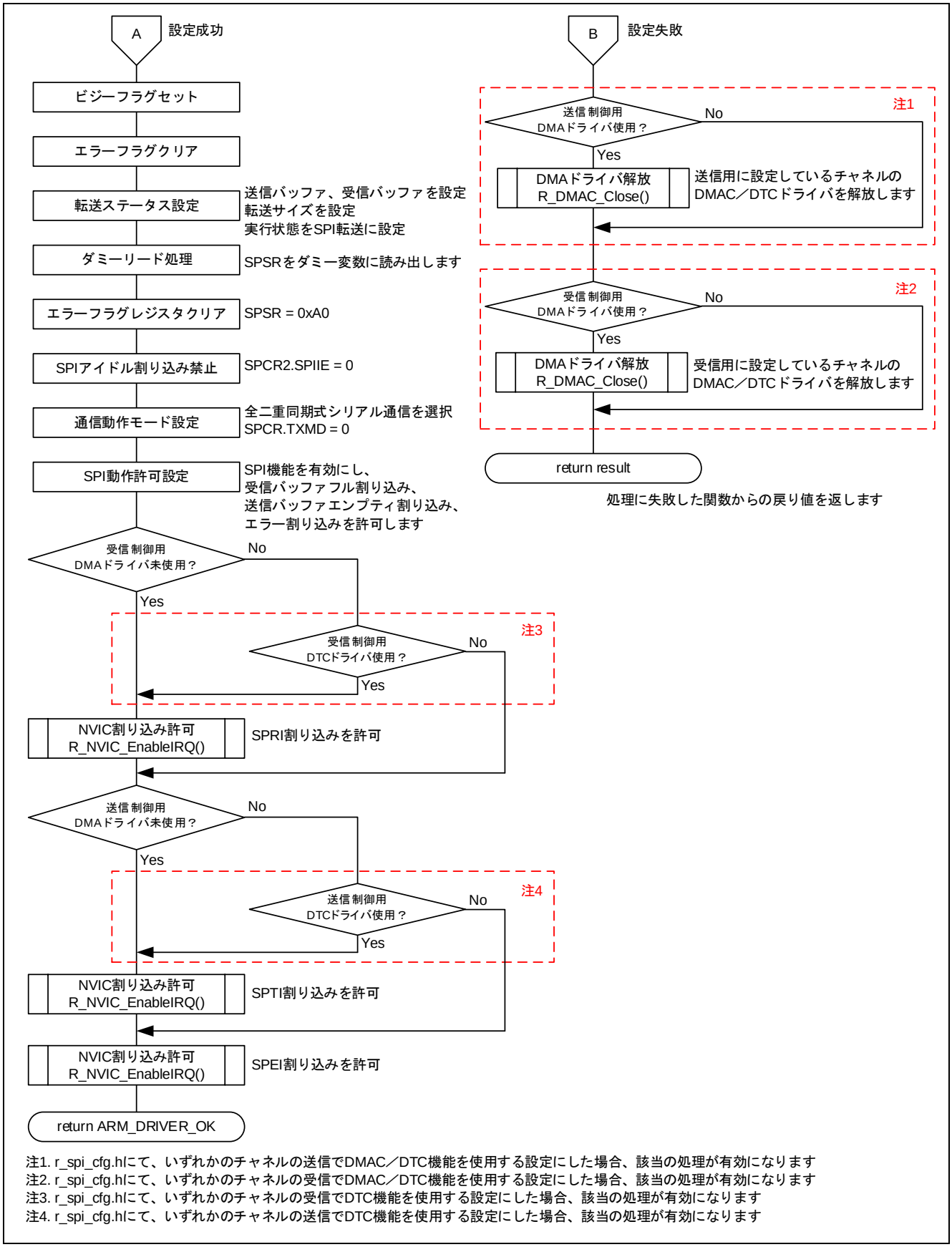


図 4-10 ARM_SPI_Transfer 関数処理フロー(1/2)



注1. r_spi_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2. r_spi_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3. r_spi_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_spi_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

図 4-11 ARM_SPI_Transfer 関数処理フロー(2/2)

4.1.9 ARM_SPI_GetDataCount 関数

表 4-9 ARM_SPI_GetDataCount 関数仕様

書式	uint32_t ARM_SPI_GetDataCount(st_spi_resources_t const * const p_spi)
仕様説明	現時点の送受信数を取得します 送信動作実行中の場合は送信済みデータ数を、受信動作もしくは送受信動作実行中の場合は受信済みデータ数を返します
引数	st_spi_resources_t * const p_spi : SPI のリソース 送受信数を取得する SPI のリソースを指定します
戻り値	送受信数

備考	インスタンスからのアクセス時は SPI 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]
	// SPI driver instance (SPI0)
	extern ARM_DRIVER_SPI Driver_SPI0;
	ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
	
	main()
	{
	uint32_t tx_count;
	tx_count = spi0Drv->GetDataCount();
	}

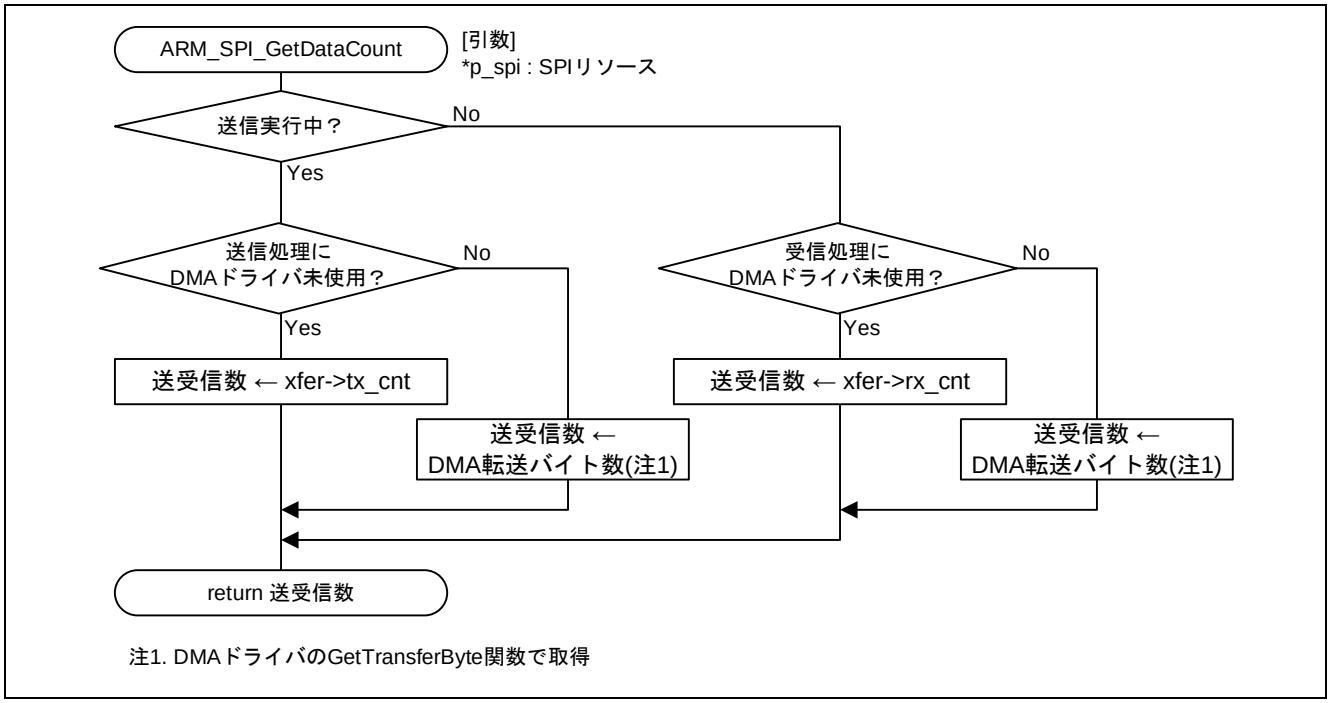


図 4-12 ARM_SPI_GetDataCount 関数処理フロー

4.1.10 ARM_SPI_Control 関数

表 4-10 ARM_SPI_Control 関数仕様

書式	int32_t ARM_SPI_Control(uint32_t control, uint32_t arg, st_spi_resources_t const * const p_spi)
仕様説明	SPI 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 SPI 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 4-11 参照）
	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI 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 ・不正なコマンドを実行した場合 ・ARM_SPI_ABORT_TRANSFER 以外のコマンド実行時に、通信動作中の場合
	ARM_DRIVER_ERROR_BUSY ビジー状態による制御コマンド実行失敗 ARM_SPI_ABORT_TRANSFER 以外のコマンド実行時に、通信動作中の場合、ビジー状態による制御コマンド実行失敗となります
	ARM_SPI_ERROR_FRAME_FORMAT フレームフォーマットエラー 以下のいずれかの状態を検出するとフレームフォーマットエラーとなります ・ARM_SPI_SS_SLAVE_SW かつ ARM_SPI_CPOL0_CPHA0 を指定した場合 ・ARM_SPI_SS_SLAVE_SW かつ ARM_SPI_CPOL1_CPHA0 を指定した場合 ・ARM_SPI_TI_SSI もしくは ARM_SPI_MICROWIRE を指定した場合
	ARM_SPI_ERROR_BIT_ORDER ビットオーダーエラー ビットオーダー設定に不正な値を設定した場合、ビットオーダーエラーとなります
	ARM_SPI_ERROR_DATA_BITS データビットエラー データビット長設定に 8、9、10、11、12、13、14、15、16、20、24、32 以外の値を指定した場合、データビットエラーとなります。
	ARM_SPI_ERROR_SS_MODE SSL 端子制御設定エラー 以下のいずれかの状態を検出すると SSL 端子制御設定エラーとなります ・ARM_SPI_MODE_MASTER コマンド実行時、SSL 制御設定に ARM_SPI_SS_MASTER_UNUSED/ARM_SPI_SS_MASTER_SW/ ARM_SPI_SS_MASTER_HW_OUTPUT 以外を指定した場合 ・ARM_SPI_MODE_SLAVE コマンド実行時、SSL 制御設定に ARM_SPI_SS_SLAVE_HW/ARM_SPI_SS_SLAVE_SW 以外を指定した場合

備考	インスタンスからのアクセス時は SPI 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]
	// SPI driver instance (SPI0)
	extern ARM_DRIVER_SPI Driver_SPI0;
	ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
	
	main()
	{
	spi0Drv->Control(ARM_SPI_ABORT_TRANSFER, NULL);
	}

表 4-11 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_SPI_MODE_INACTIVE	NULL(0)	引数は使用しません
ARM_SPI_MODE_MASTER	ボーレート (MAX : PCLKA/2 MIN:PCLKA/4096)	指定されたボーレートでマスタモードを初期化します
ARM_SPI_MODE_SLAVE	NULL(0)	引数は使用しません
ARM_SPI_SET_BUS_SPEED	ボーレート (MAX : PCLKA/2 MIN:PCLKA/4096)	転送速度を設定します 第 2 引数にはボーレートを指定してください
ARM_SPI_GET_BUS_SPEED	NULL(0)	引数は使用しません
ARM_SPI_SET_DEFAULT_TX_VALUE	デフォルトデータ (0x00～0xFFFFFFFF) (注)	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_SPI_CONTROL_SS	ARM_SPI_SS_INACTIVE	SSL0 端子を非アクティブ("H")状態にします
	ARM_SPI_SS_ACTIVE	SSL0 端子をアクティブ("L")状態にします
ARM_SPI_ABORT_TRANSFER	NULL(0)	引数は使用しません

注　デフォルトデータの最大データサイズは、データビット長設定によって異なります。

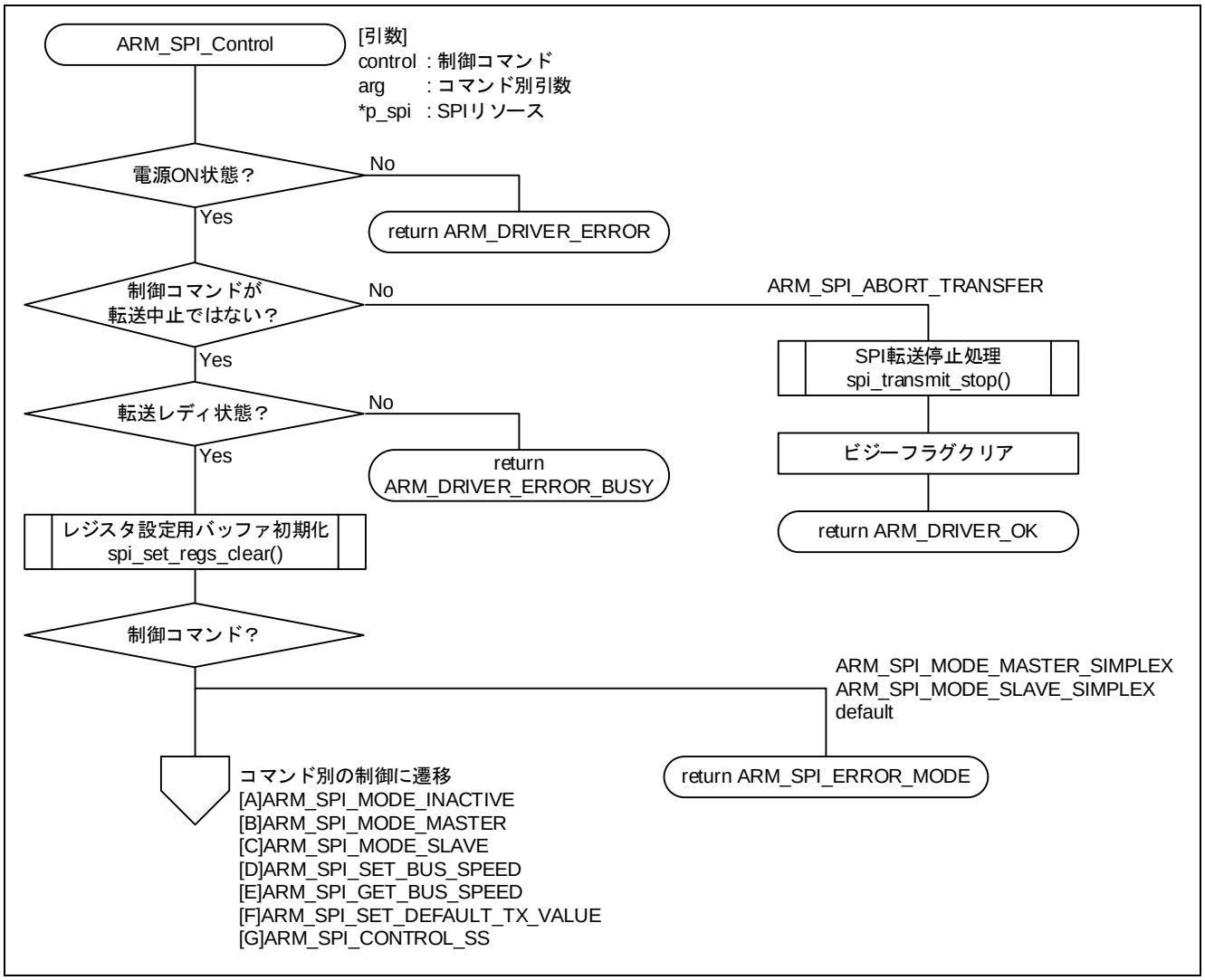


図 4-13 ARM_SPI_Control 関数処理フロー(1/3)

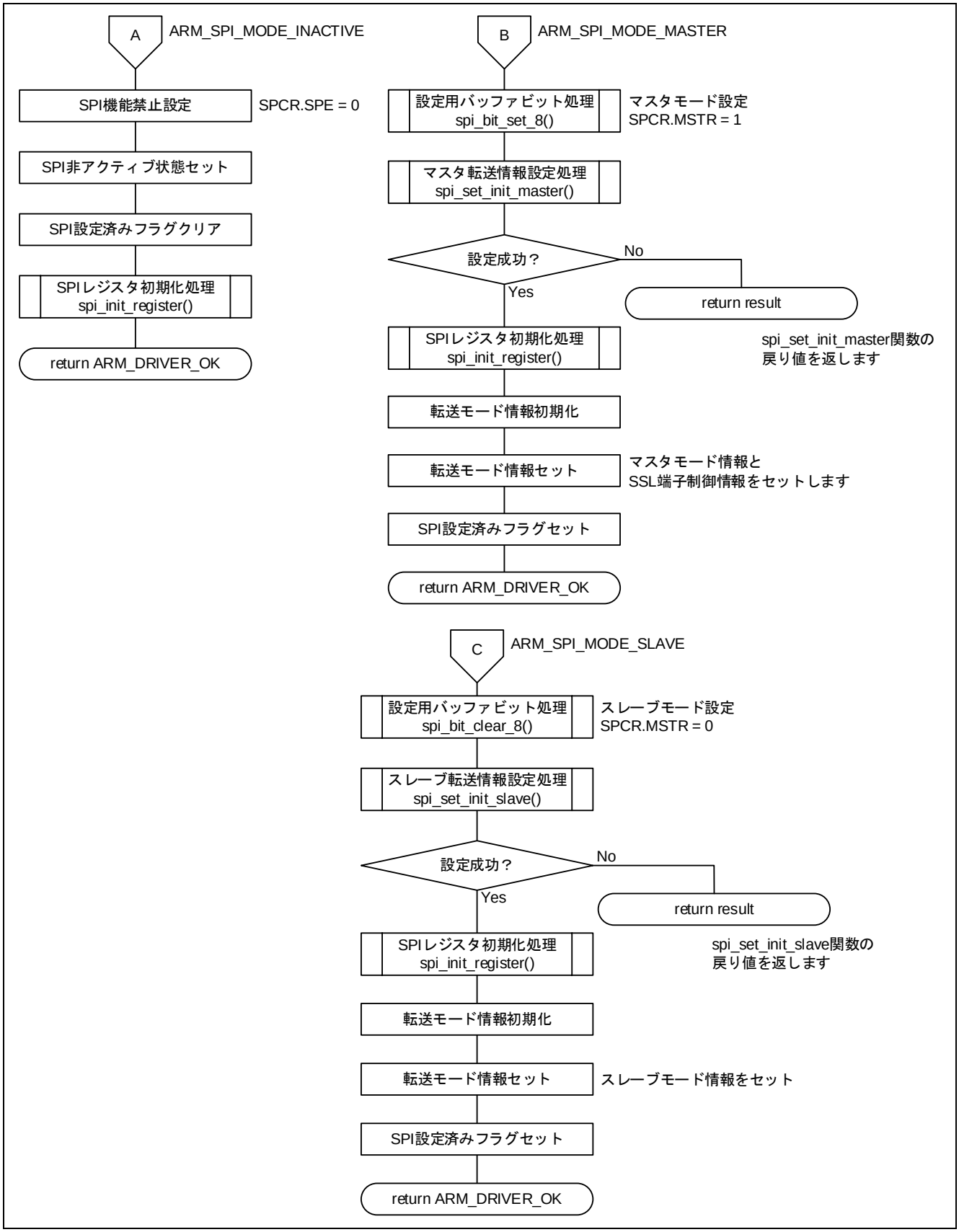


図 4-14 ARM_SPI_Control 関数処理フロー(2/3)

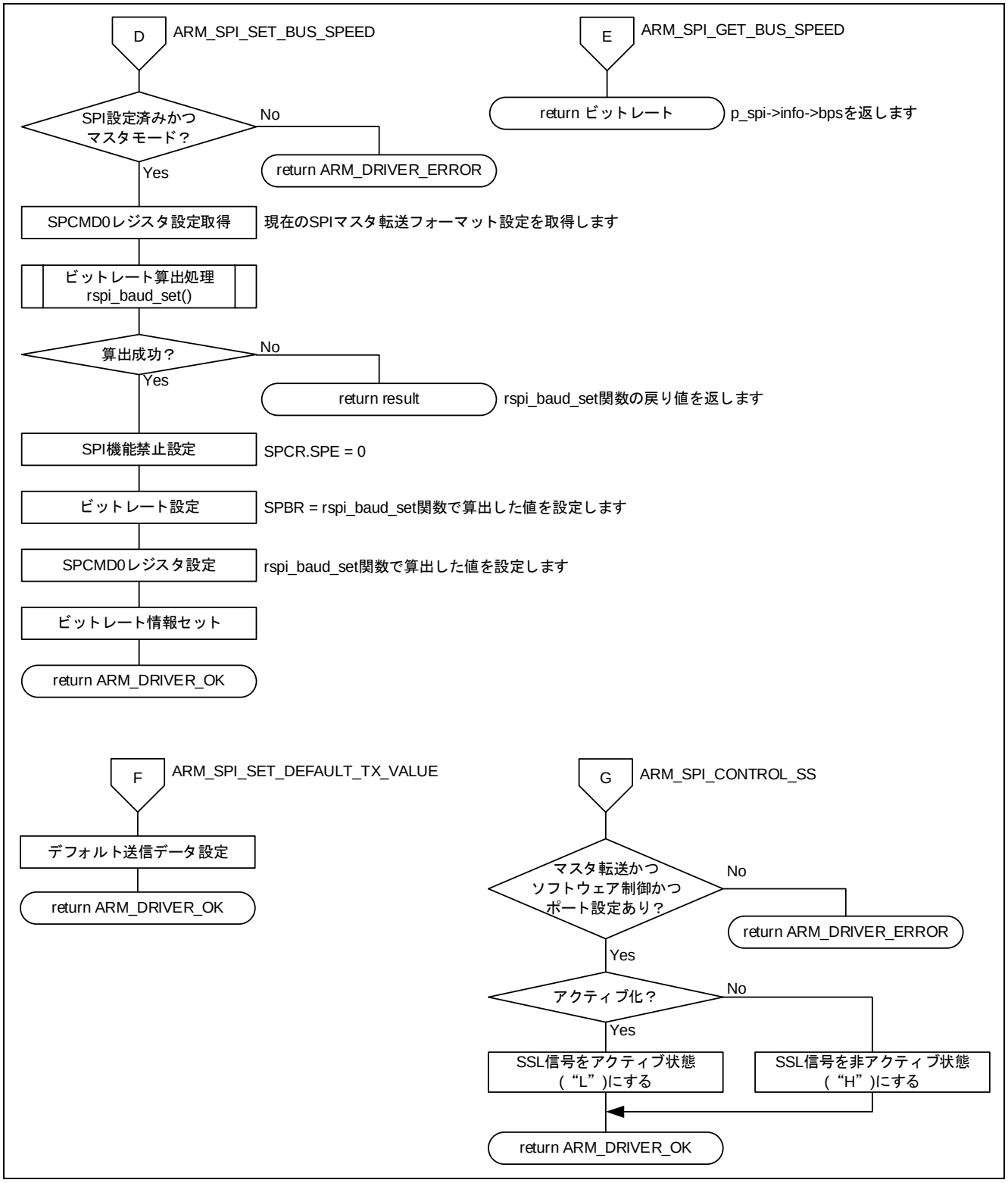


図 4-15 ARM_SPI_Control 関数処理フロー(3/3)

4.1.11 ARM_SPI_GetStatus 関数

表 4-12 ARM_SPI_GetStatus 関数仕様

書式	ARM_SPI_STATUS ARM_SPI_GetStatus(st_spi_resources_t const * const p_spi)
仕様説明	SPI のステータスを返します
引数	st_spi_resources_t * const p_spi : SPI のリソース 対象の SPI 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 SPI 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]
	// SPI driver instance (SPI0)
	extern ARM_DRIVER_SPI Driver_SPI0;
	ARM_DRIVER_SPI *spi0Drv = &Driver_SPI0;
	
	main()
	{
	ARM_SPI_STATUS state;
	state = spi0Drv->GetStatus();
	
	}

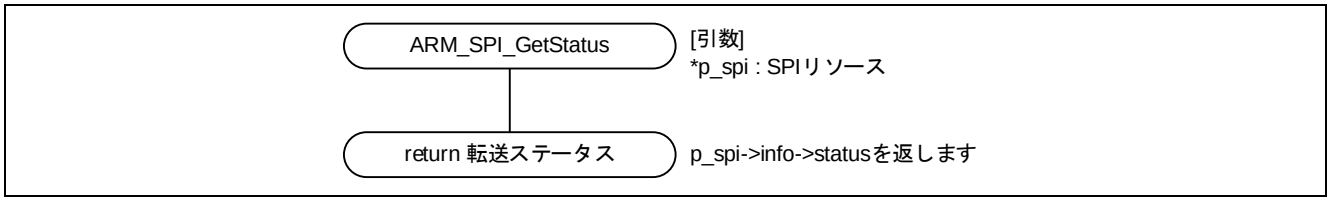


図 4-16 ARM_SPI_GetStatus 関数処理フロー

4.1.12 spi_set_init_master 関数

表 4-13 spi_set_init_master 関数仕様

書式	static int32_t spi_set_init_master(uint32_t control, uint32_t baudrate, st_spi_reg_buf_t * const p_spi_regs)
仕様説明	マスタモードの設定を行います
引数	uint32_t control : 制御コマンド
	uint32_t baudrate : ボーレート設定
	st_spi_reg_set_t * const p_spi_regs : レジスタ設定値格納ポインタ
戻り値	ARM_DRIVER_OK　マスタモード設定成功
	ARM_DRIVER_ERROR　マスタモード設定失敗 ボーレート設定値が不正（PCLKA の分周比 2～4096 範囲外）な場合、マスタモード設定失敗となります
	ARM_SPI_ERROR_SS_MODE　SSL 端子制御設定エラー SSL 制御設定に ARM_SPI_SS_MASTER_UNUSED/ARM_SPI_SS_MASTER_SW/ ARM_SPI_SS_MASTER_HW_OUTPUT 以外を指定した場合、SSL 端子制御設定エラーとなります
	ARM_SPI_ERROR_FRAME_FORMAT　フレームフォーマットエラー ARM_SPI_TI_SSI もしくは ARM_SPI_MICROWIRE を指定した場合、フレームフォーマットエラーとなります
	ARM_SPI_ERROR_BIT_ORDER　ビットオーダーエラー ビットオーダー設定に不正な値を設定した場合、ビットオーダーエラーとなります
	ARM_SPI_ERROR_DATA_BITS　データビットエラー データビット長設定に 8、9、10、11、12、13、14、15、16、20、24、32 以外の値を指定した場合、データビットエラーとなります。
備考	-

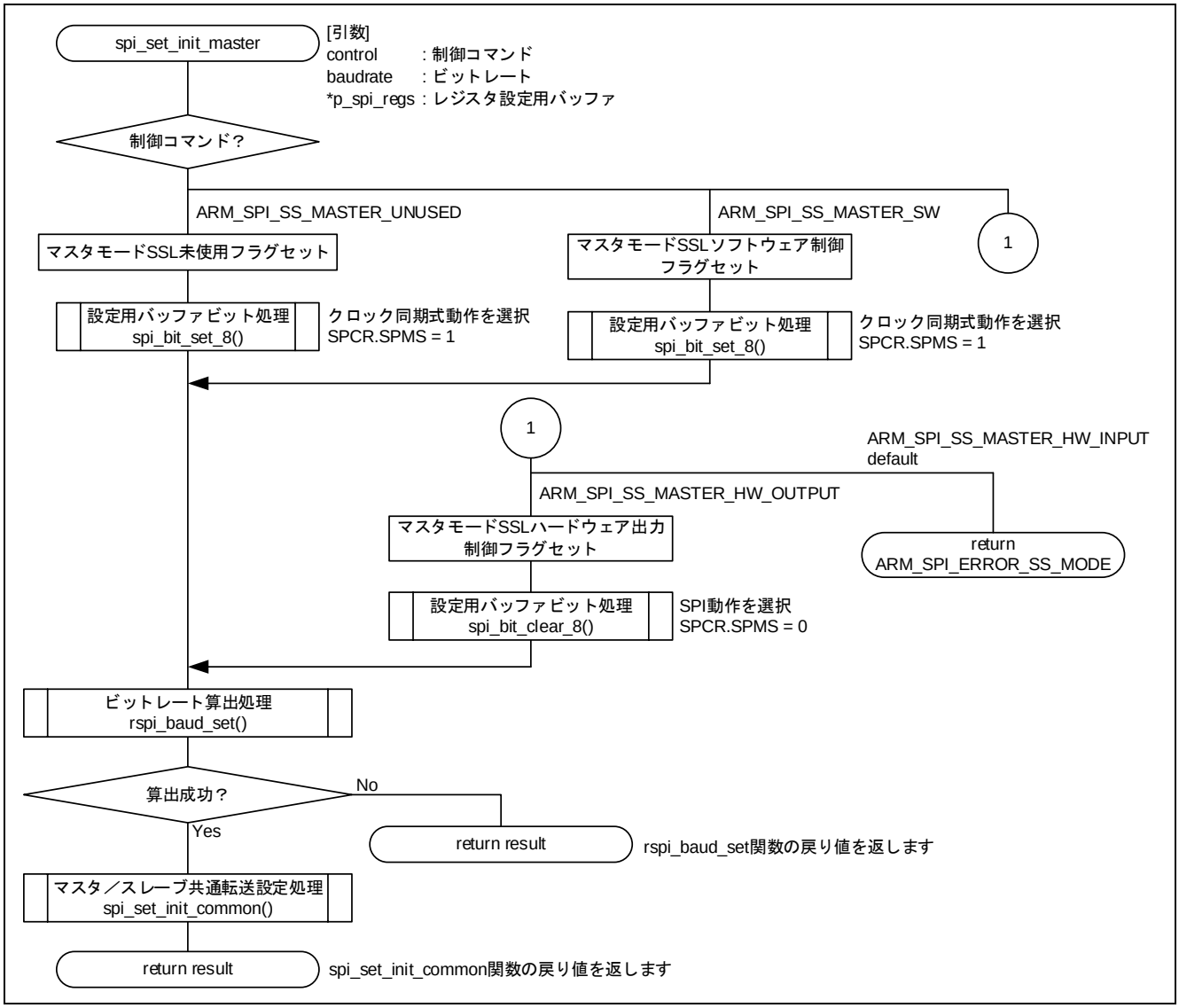


図 4-17 spi_set_init_master 関数処理フロー

4.1.13 spi_set_init_slave 関数

表 4-14 spi_set_init_slave 関数仕様

書式	static int32_t spi_set_init_slave(uint32_t control, st_spi_reg_buf_t * const p_spi_regs)
仕様説明	スレーブモードの設定を行います
引数	uint32_t control : 制御コマンド
	st_spi_reg_set_t * const p_spi_regs : レジスタ設定値格納ポインタ
戻り値	ARM_DRIVER_OK　　スレーブモード設定成功
	ARM_SPI_ERROR_SS_MODE　　SSL 端子制御設定エラー SSL 制御設定に ARM_SPI_SS_SLAVE_HW/ARM_SPI_SS_SLAVE_SW 以外を指定した場合、SSL 端子制御設定エラーとなります
備考	-

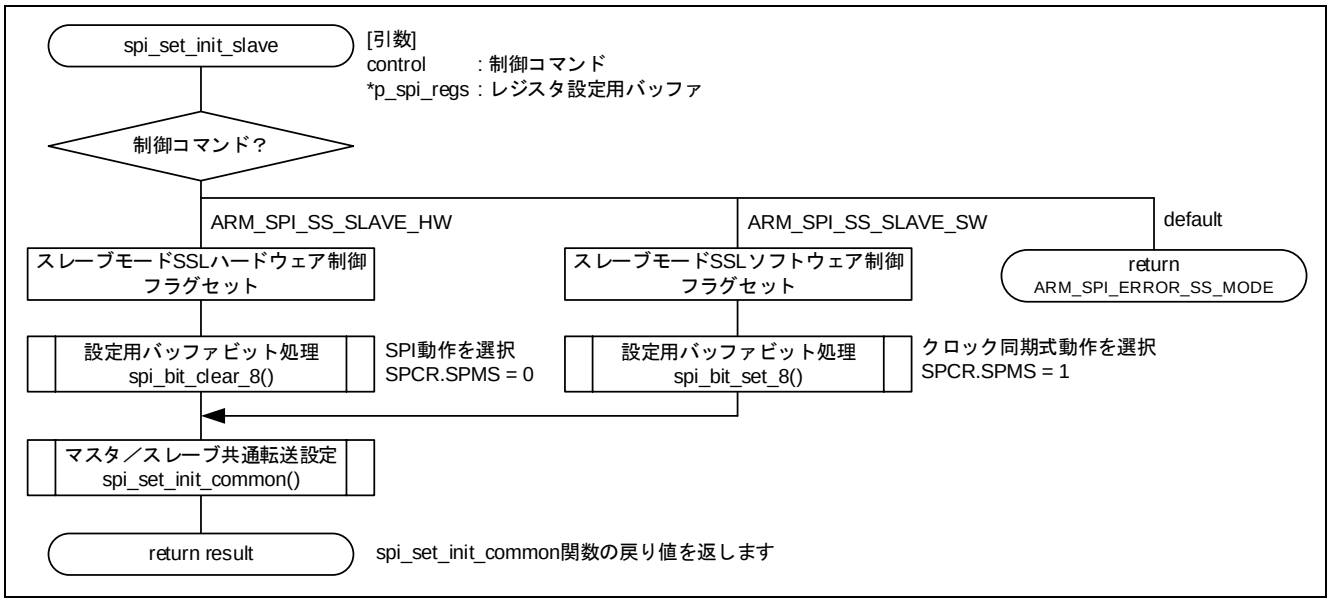


図 4-18 spi_set_init_slave 関数処理フロー

4.1.14 spi_set_init_common 関数

表 4-15 spi_set_init_common 関数仕様

書式	static int32_t spi_set_init_common(uint32_t control, st_spi_reg_buf_t * const p_spi_regs)
仕様説明	マスタ/スレーブモード共通の転送設定を行います
引数	uint32_t control : 制御コマンド
	st_spi_reg_set_t * const p_spi_regs : レジスタ設定値格納ポインタ
戻り値	ARM_DRIVER_OK　マスタ/スレーブモード共通設定成功
	ARM_SPI_ERROR_FRAME_FORMAT　フレームフォーマットエラー 以下のいずれかの状態を検出するとフレームフォーマットエラーとなります ・ARM_SPI_SS_SLAVE_SW かつ ARM_SPI_CPOL0_CPHA0 を指定した場合 ・ARM_SPI_SS_SLAVE_SW かつ ARM_SPI_CPOL1_CPHA0 を指定した場合 ・ARM_SPI_TI_SSI もしくは ARM_SPI_MICROWIRE を指定した場合
	ARM_SPI_ERROR_BIT_ORDER　ビットオーダーエラー ビットオーダー設定に不正な値を設定した場合、ビットオーダーエラーとなります
	ARM_SPI_ERROR_DATA_BITS　データビットエラー データビット長設定に 8、9、10、11、12、13、14、15、16、20、24、32 以外の値を指定した場合、データビットエラーとなります。
備考	-

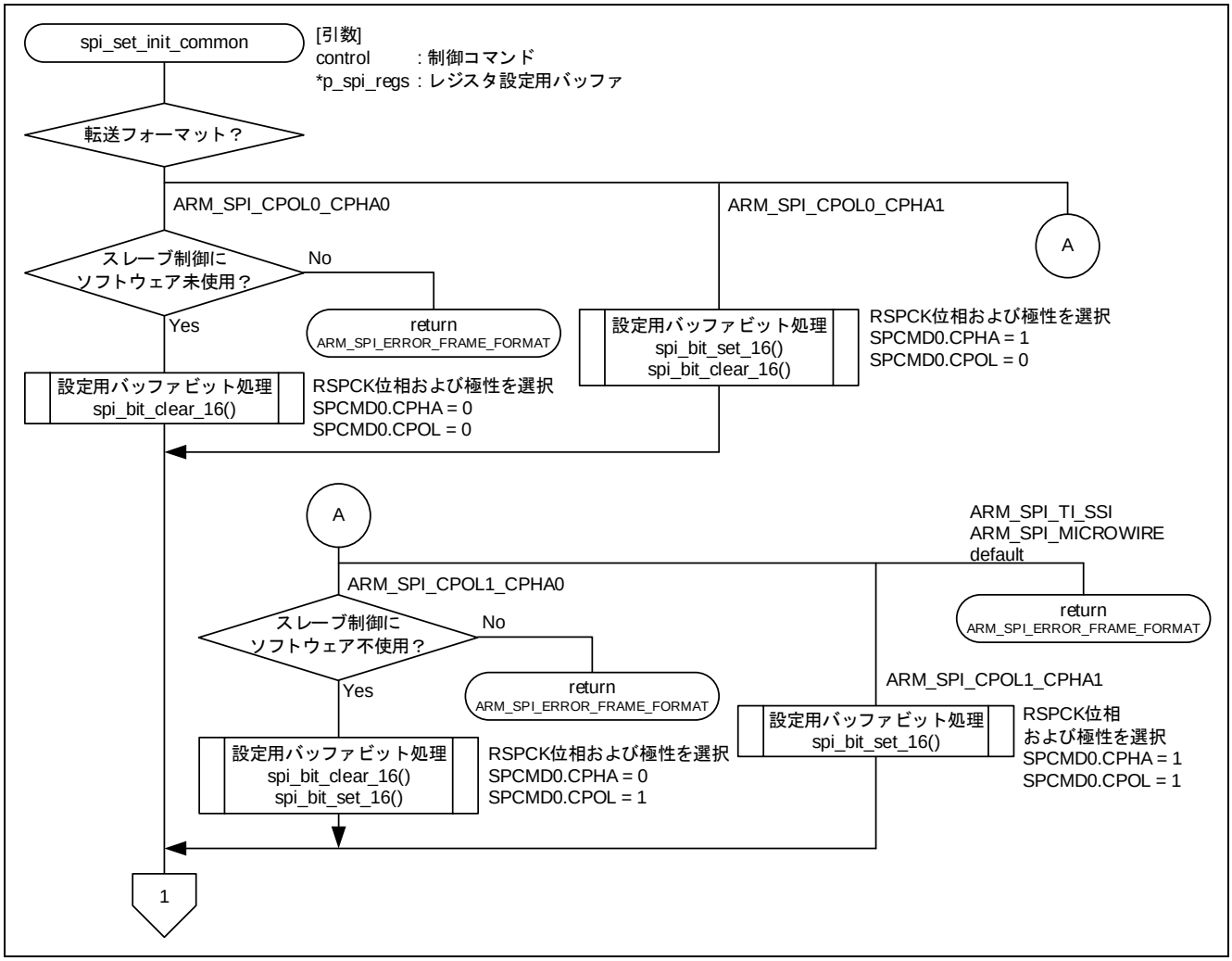


図 4-19 spi_set_init_common 関数処理フロー(1/2)

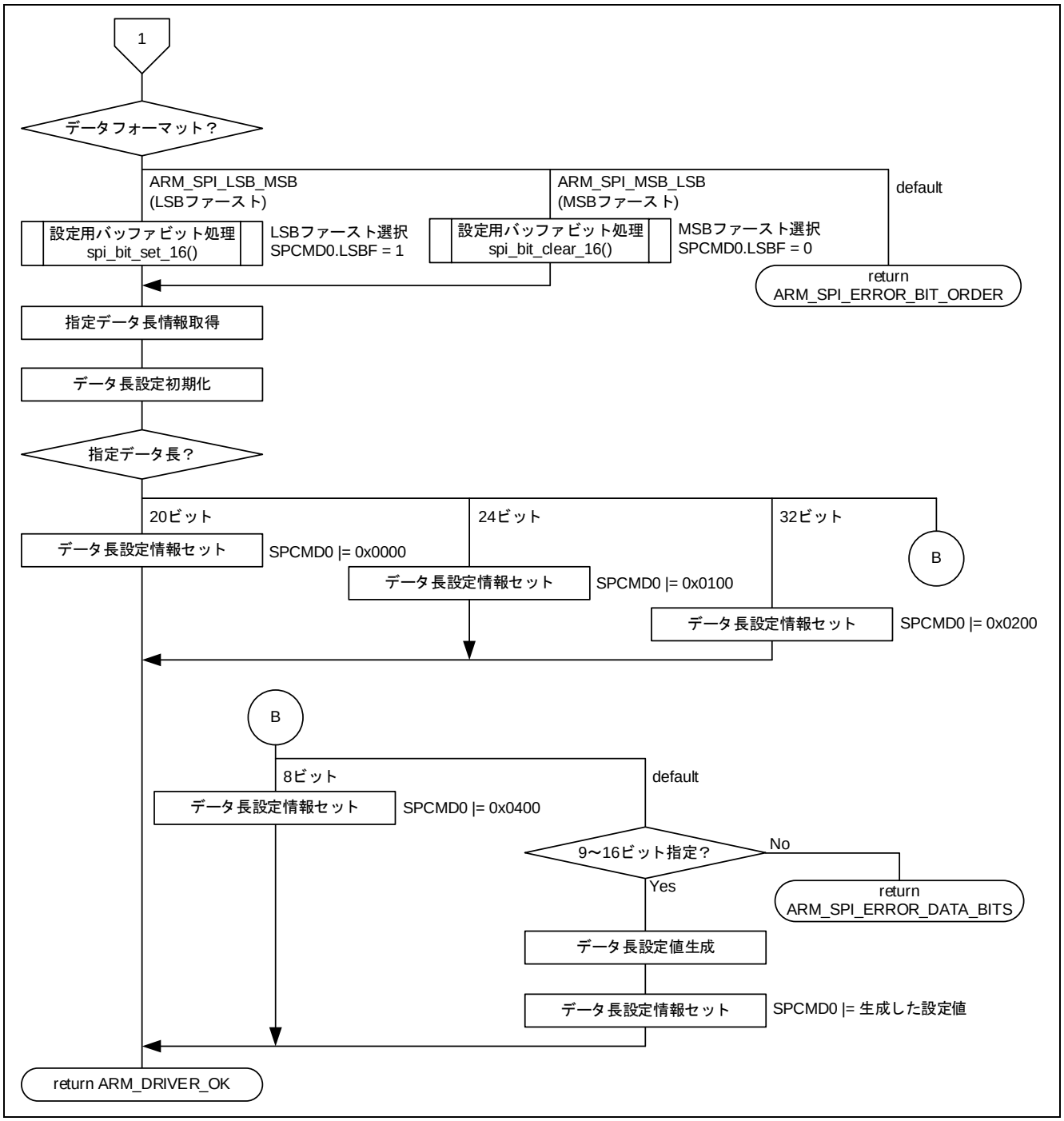


図 4-20 spi_set_init_common 関数処理フロー(2/2)

4.1.15 rspi_baud_set 関数

表 4-16 rspi_baud_set 関数仕様

書式	static int32_t rspi_baud_set(st_spi_reg_buf_t * const p_spi_regs, uint32_t bps_target)
仕様説明	ボーレートの算出を行います
引数	st_spi_reg_buf_t * const p_spi_regs : レジスタ設定用バッファポインタ ボーレートの算出結果を格納するバッファポインタ
	uint32_t bps_target : ボーレート設定値
戻り値	ARM_DRIVER_OK　　ボーレート算出成功
	ARM_DRIVER_ERROR　　ボーレート算出失敗 ボーレート設定値が不正（PCLKA の分周比 2～4096 範囲外）な場合、ボーレート設定値エラーとなります
備考	-

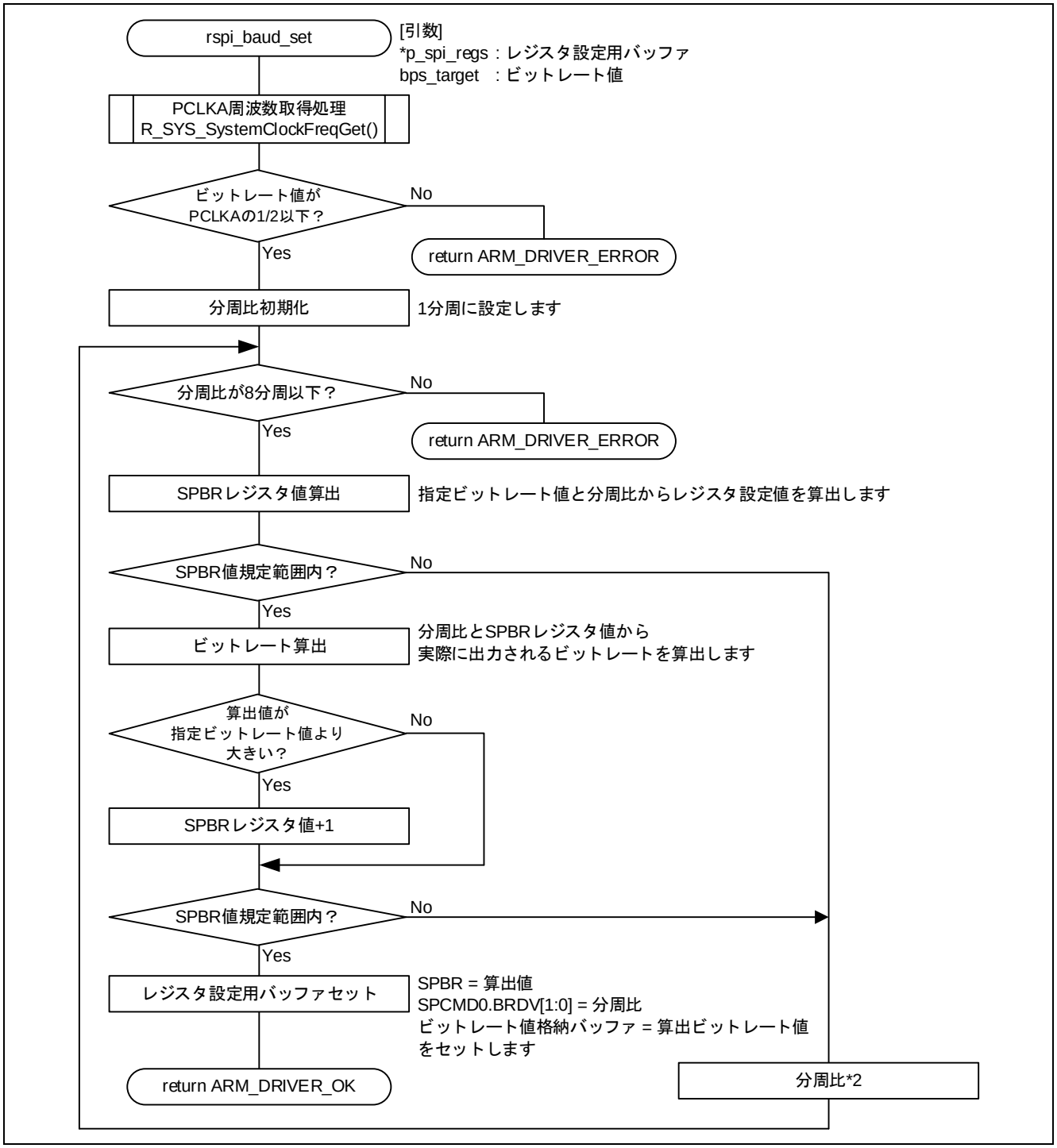


図 4-21 rspi_baud_set 関数処理フロー

4.1.16 spi_set_regs_clear 関数

表 4-17 spi_set_regs_clear 関数仕様

書式	static void spi_set_regs_clear(st_spi_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_spi_reg_buf_t * const p_spi_regs : レジスタ設定用バッファポインタ
戻り値	なし
備考	-

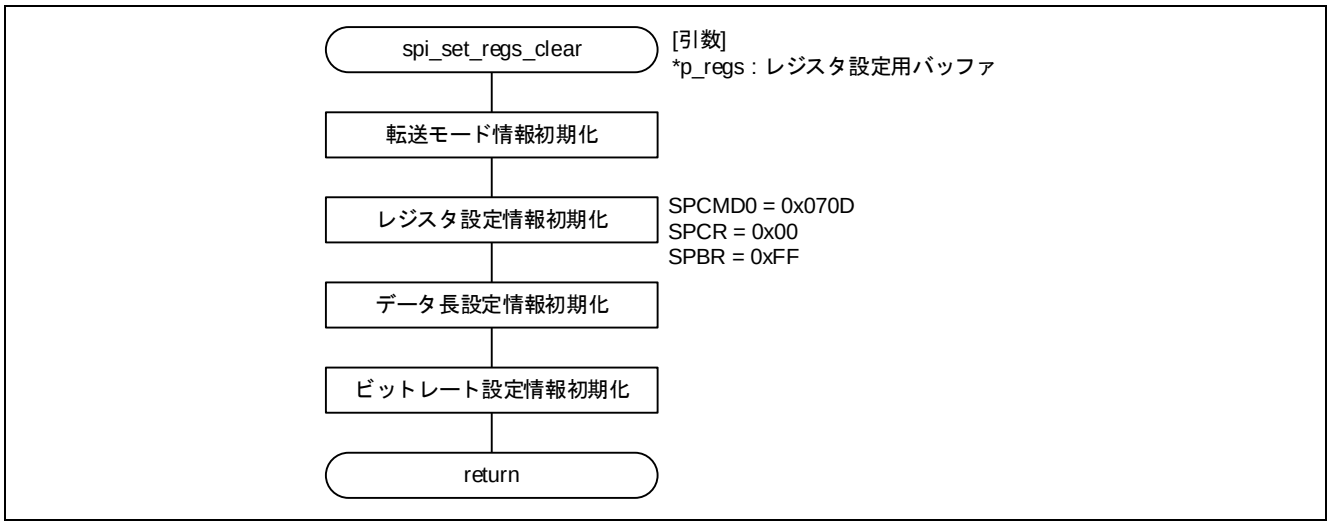


図 4-22 spi_set_regs_clear 関数処理フロー

4.1.17 spi_init_register 関数

表 4-18 spi_init_register 関数仕様

書式	static void spi_init_register(st_spi_reg_buf_t const * const p_spi_regs, st_spi_resources_t * const p_spi)
仕様説明	SPI 機能設定レジスタを初期化します
引数	st_spi_reg_buf_t * const p_spi_regs : レジスタ設定用バッファポインタ
	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI のリソースを指定します
戻り値	なし
備考	–

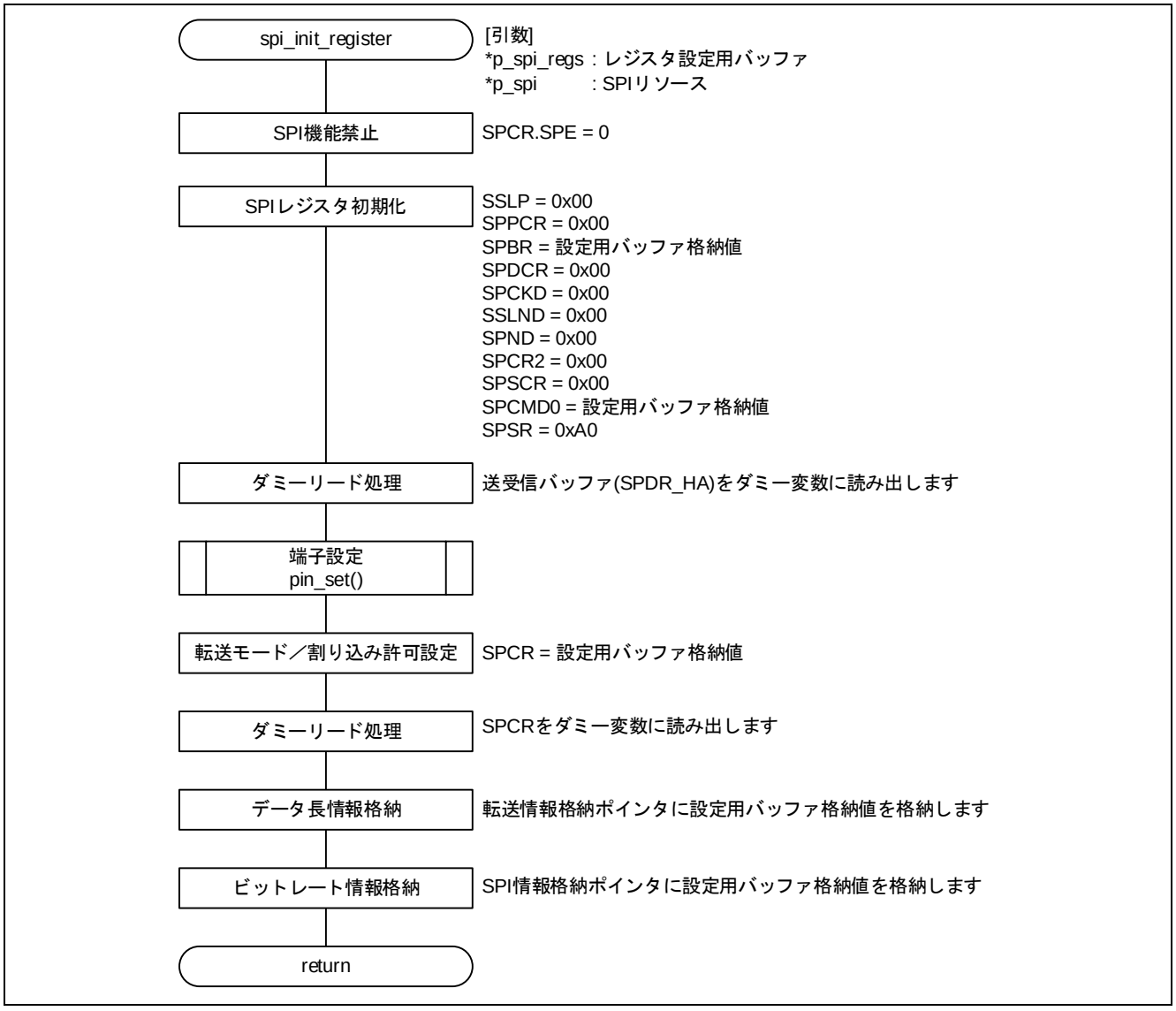


図 4-23 spi_init_register 関数処理フロー

4.1.18 spi_interrupt_disable 関数

表 4-19 spi_interrupt_disable 関数仕様

書式	static void spi_interrupt_disable(st_spi_resources_t * const p_spi)
仕様説明	割り込み禁止設定を行います
引数	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI のリソースを指定します
戻り値	なし
備考	–

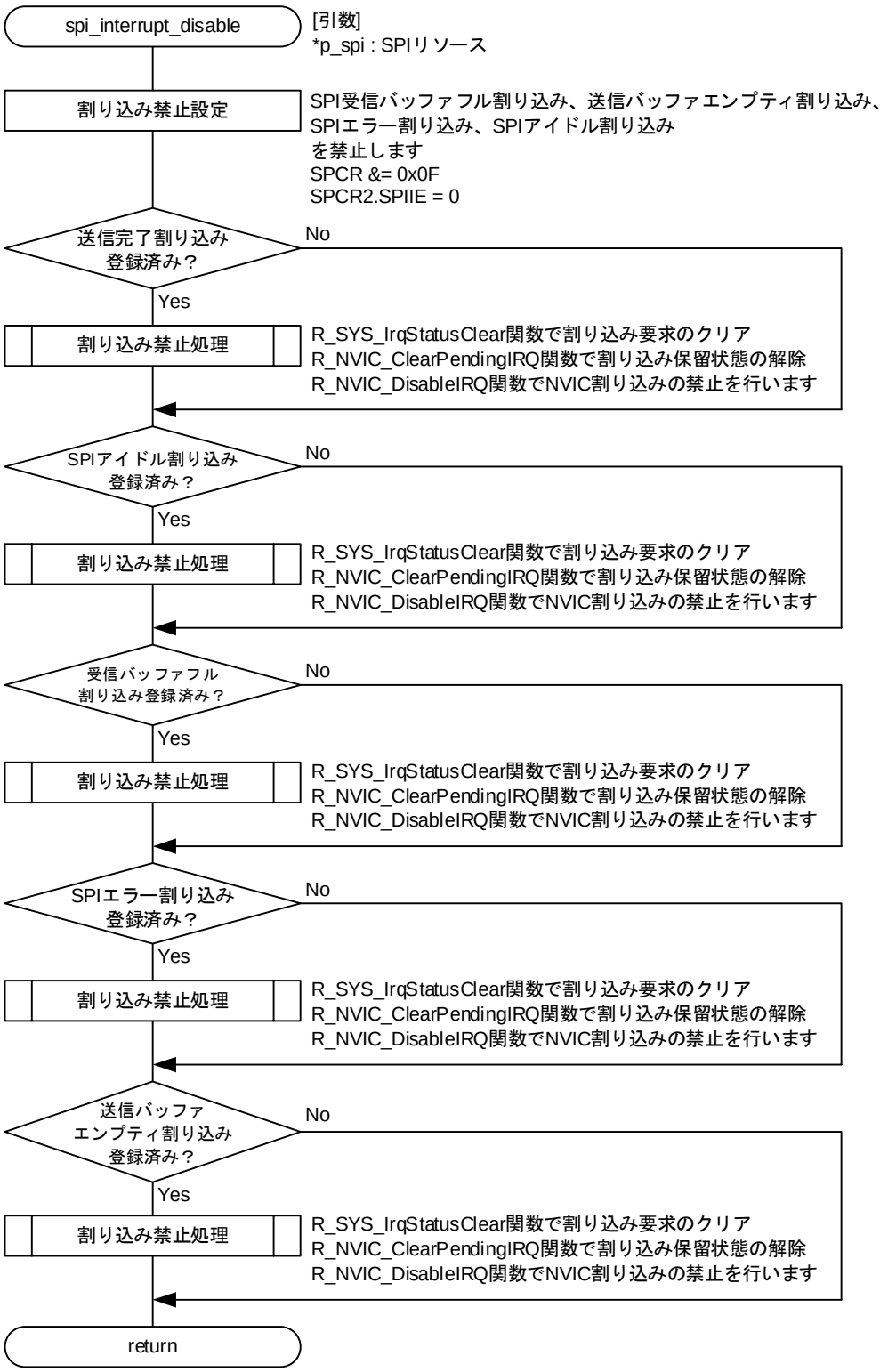


図 4-24 spi_interrupt_disable 関数処理フロー

4.1.19 spi_ir_flag_clear 関数

表 4-20 spi_ir_flag_clear 関数仕様

書式	static int32_t spi_ir_flag_clear(st_spi_resources_t * const p_spi)
仕様説明	送信/受信割り込みの IR を 0 クリアします
引数	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI のリソースを指定します
戻り値	ARM_DRIVER_OK　　IR フラグクリア成功
	ARM_DRIVER_ERROR_TIMEOUT　　タイムアウトエラー 以下のいずれかの状態を検出するとタイムアウトエラーとなります ・SPCR.SPRIE ビットの 0 クリア待ち処理でタイムアウトが発生した場合 ・SPCR.SPTIE ビットの 0 クリア待ち処理でタイムアウトが発生した場合
備考	–

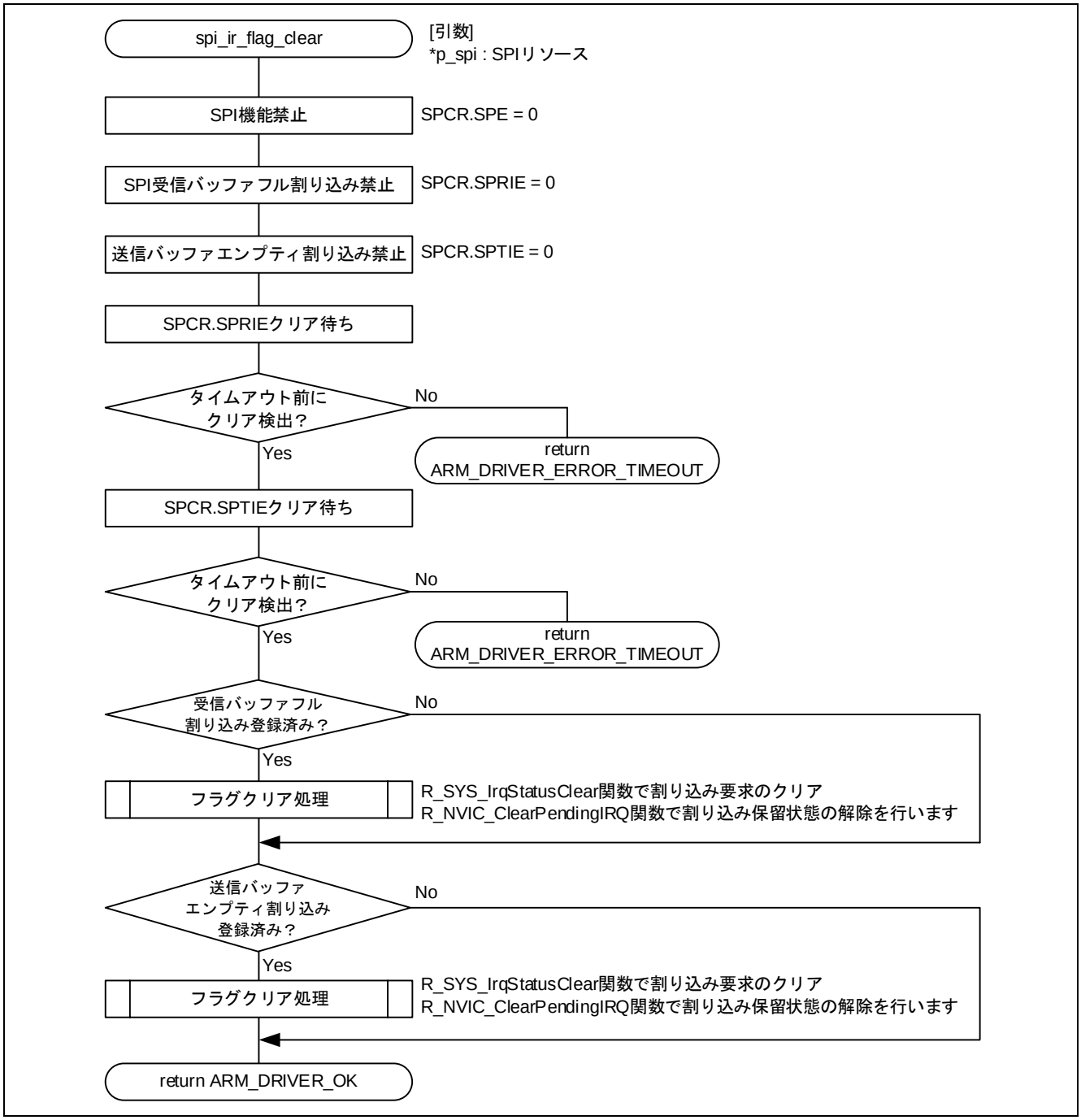
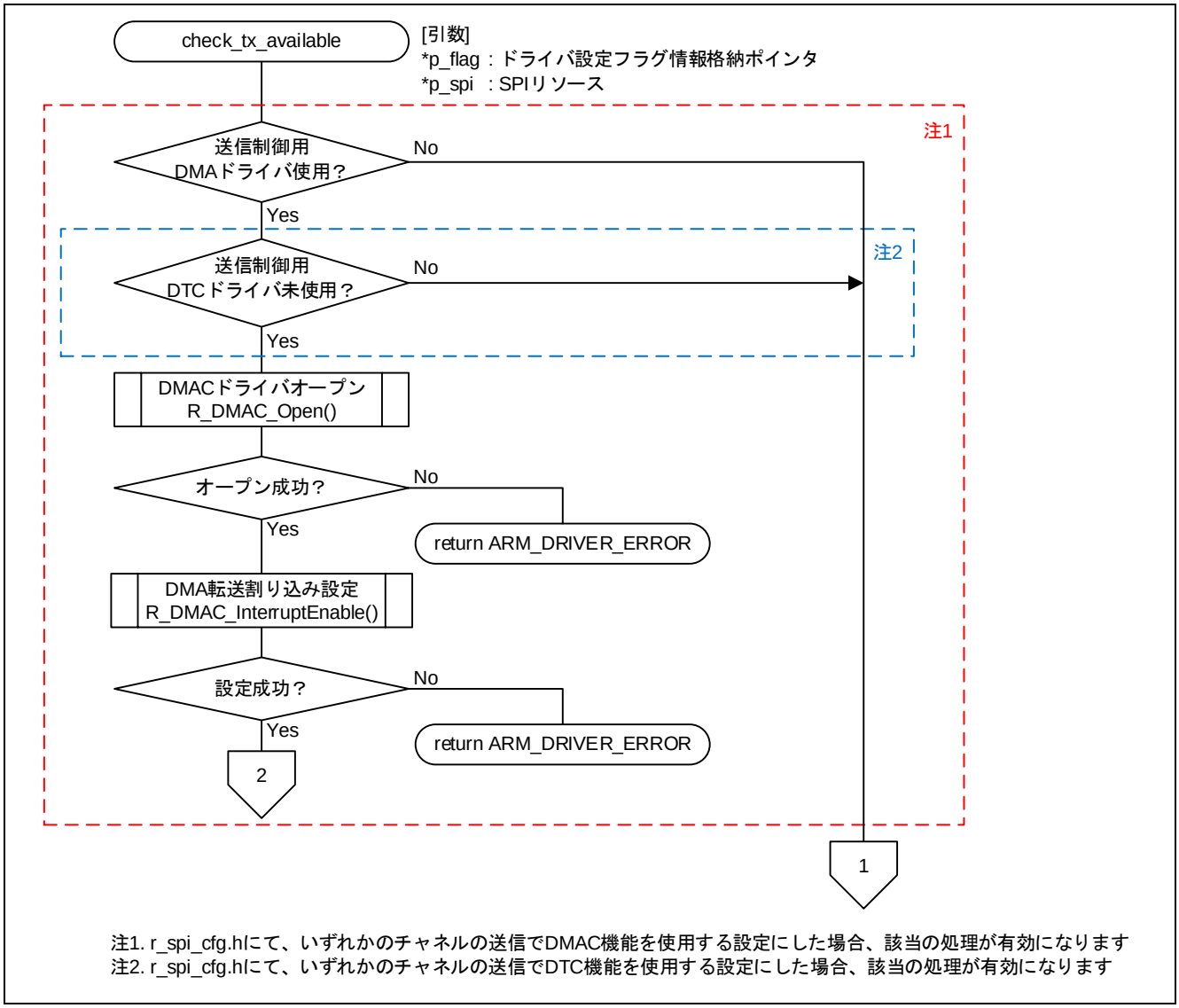


図 4-25 spi_ir_flag_clear 関数処理フロー

4.1.20 check_tx_available 関数

表 4-21 check_tx_available 関数仕様

書式	static int32_t check_tx_available(int16_t * const p_flag, st_spi_resources_t * const p_spi)
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_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI 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、SPTI 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 SPTI 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 ・SPII 割り込みを r_system_cfg.h で NVIC に登録した状態で、SPII 割り込みのイベントリング設定に失敗した場合 ・SPII 割り込みを r_system_cfg.h で NVIC に登録した状態で、SPII 割り込みの割り込み優先レベル設定に失敗した場合 ・SPTEND、SPEI 割り込みを r_system_cfg.h で NVIC に登録した状態で、SPTEND、SPEI 割り込みのイベントリング設定に失敗した場合 ・SPTEND、SPEI 割り込みを r_system_cfg.h で NVIC に登録した状態で、SPTEND、SPEI 割り込みの割り込み優先レベル設定に失敗した場合
備考	-



注1. r_spi_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
注2. r_spi_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

図 4-26 check_tx_available 関数処理フロー(1/2)

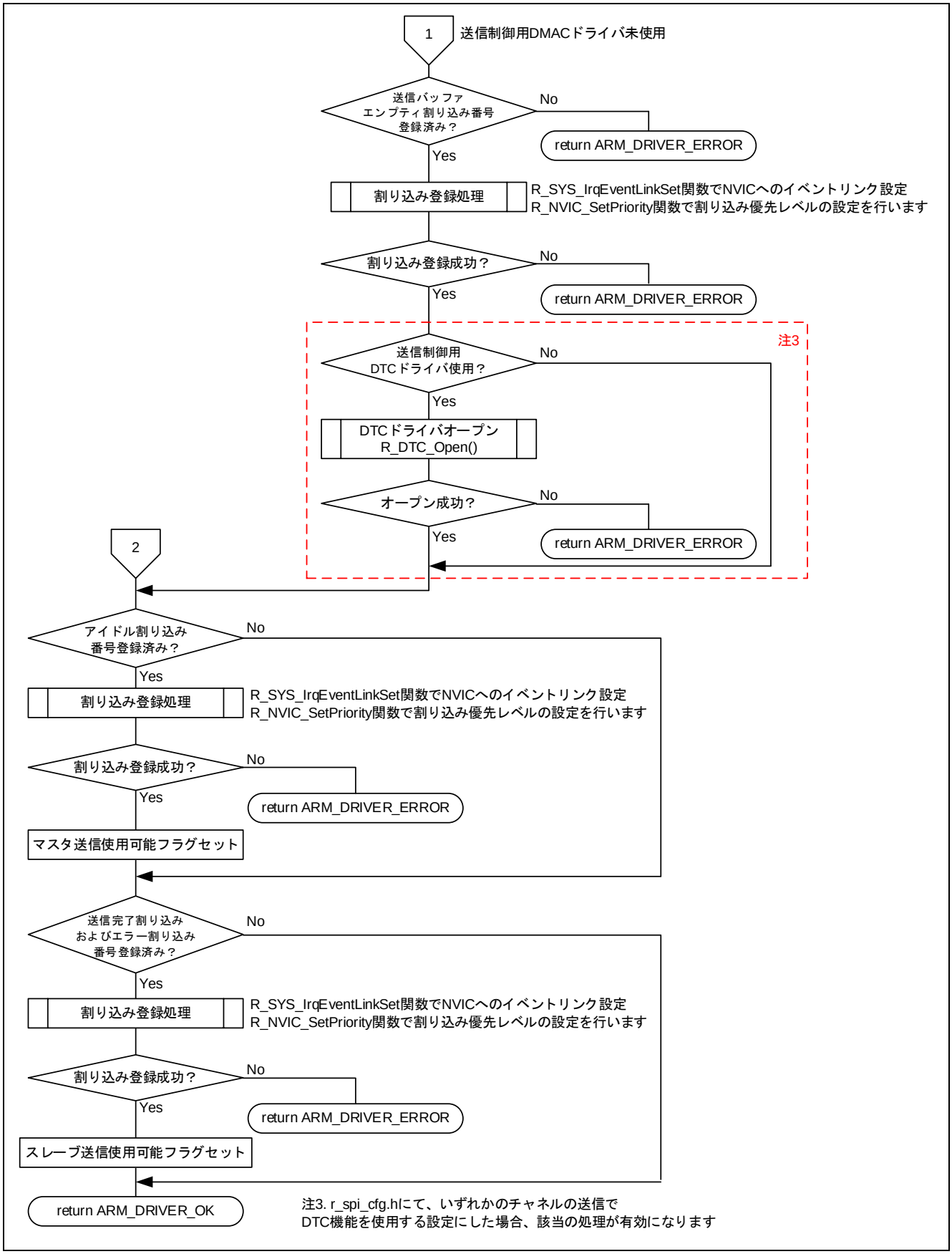
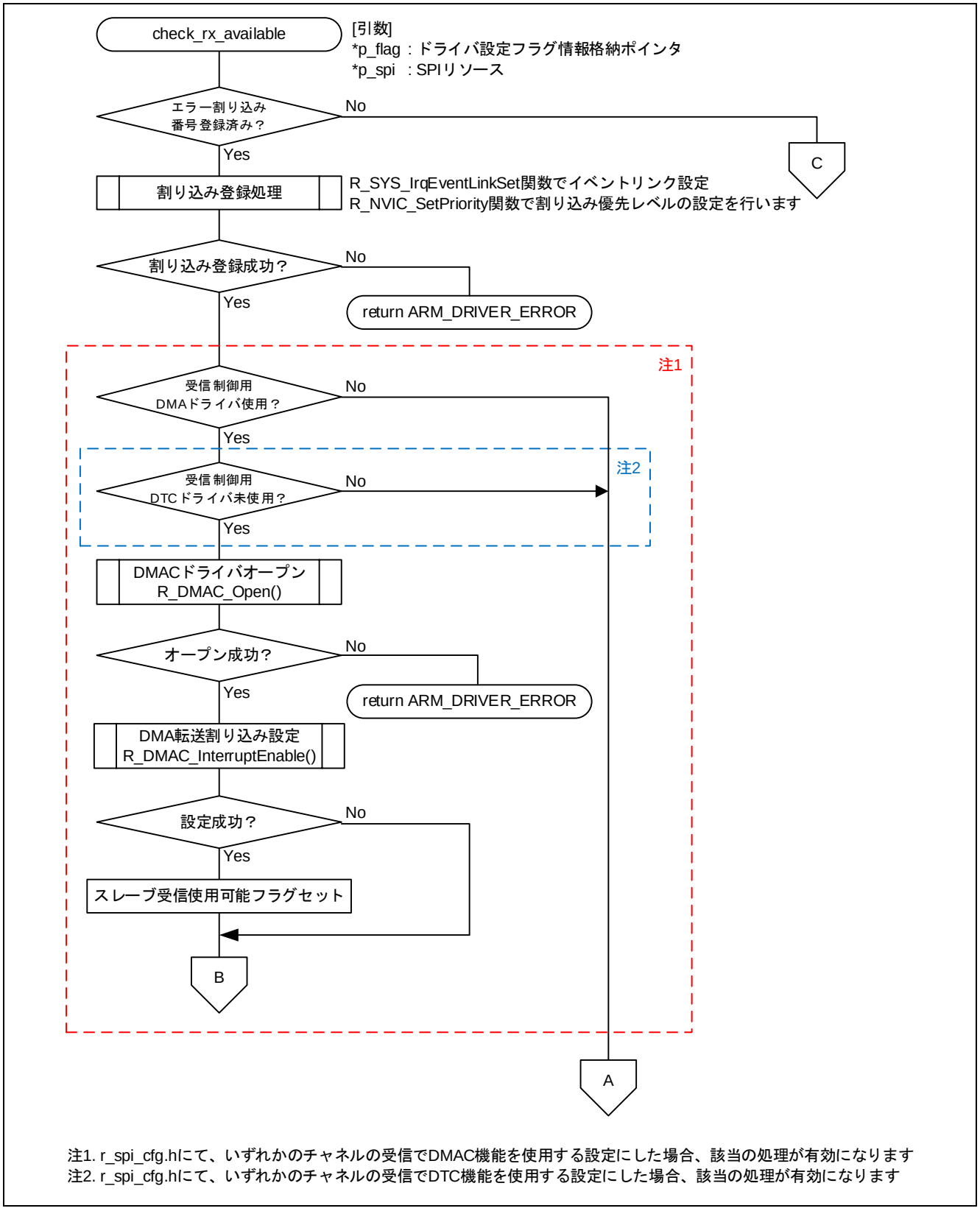


図 4-27 check_tx_available 関数処理フロー(2/2)

4.1.21 check_rx_available 関数

表 4-22 check_rx_available 関数仕様

書式	static int32_t check_rx_available(int16_t * const p_flag, st_spi_resources_t * const p_spi)
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_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI 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 ・SPRI、SPEI 割り込みのイベントリンク設定に失敗した場合 ・SPRI、SPEI 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 SPRI、SPEI 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	-



注1. r_spi_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
注2. r_spi_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

図 4-28 check_rx_available 関数処理フロー(1/2)

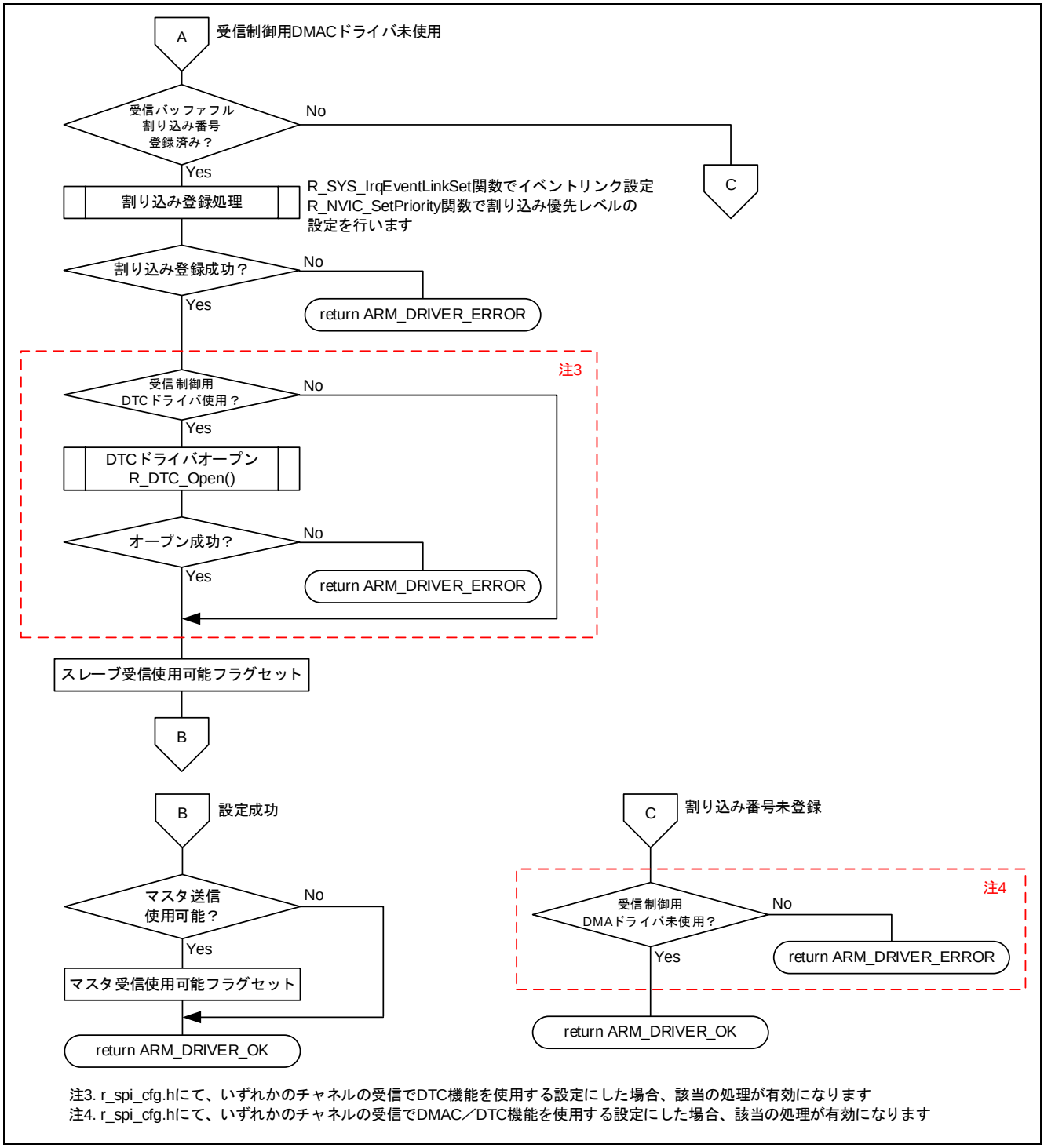
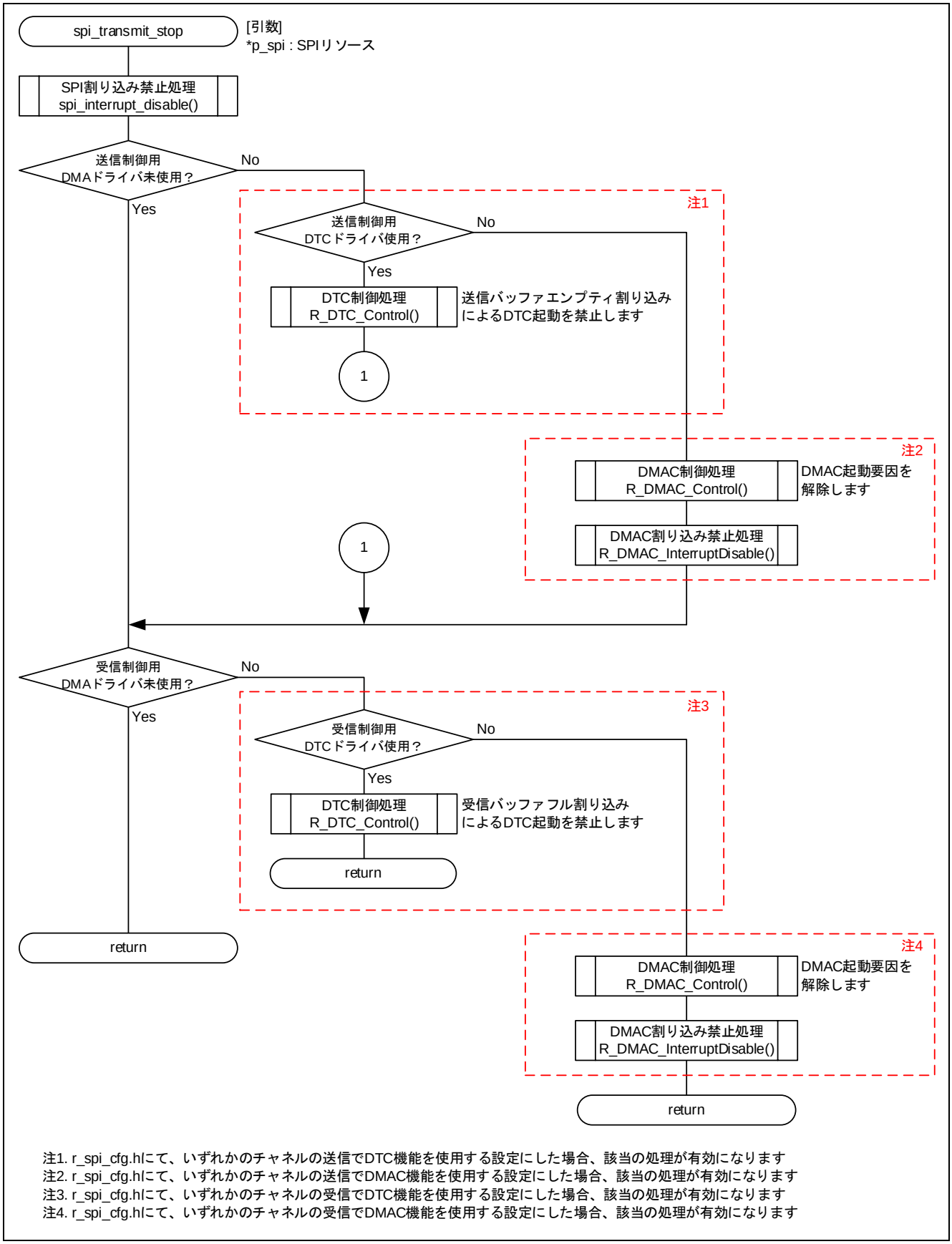


図 4-29 check_rx_available 関数処理フロー(2/2)

4.1.22 spi_transmit_stop 関数

表 4-23 spi_transmit_stop 関数仕様

書式	static void spi_transmit_stop(st_spi_resources_t const * const p_spi)
仕様説明	送信を中断します
引数	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI のリソースを指定します
戻り値	なし
備考	-



注1. r_spi_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_spi_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
注3. r_spi_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_spi_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 4-30 spi_transmit_stop 関数処理フロー

4.1.23 spi_send_setting 関数

表 4-24 spi_send_setting 関数仕様

書式	static int32_t spi_send_setting(void const * const p_data, uint32_t num, bool dummy_flag, st_spi_resources_t * const p_spi)
仕様説明	送信設定を行います
引数	void const * const p_data : 送信データ格納ポインタ
	uint32_t num : 送信サイズ
	bool dummy_flag : ダミー送信フラグ
	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI のリソースを指定します
戻り値	ARM_DRIVER_OK 送信設定成功
	ARM_DRIVER_ERROR 送信設定失敗 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、送信設定失敗となります
備考	-

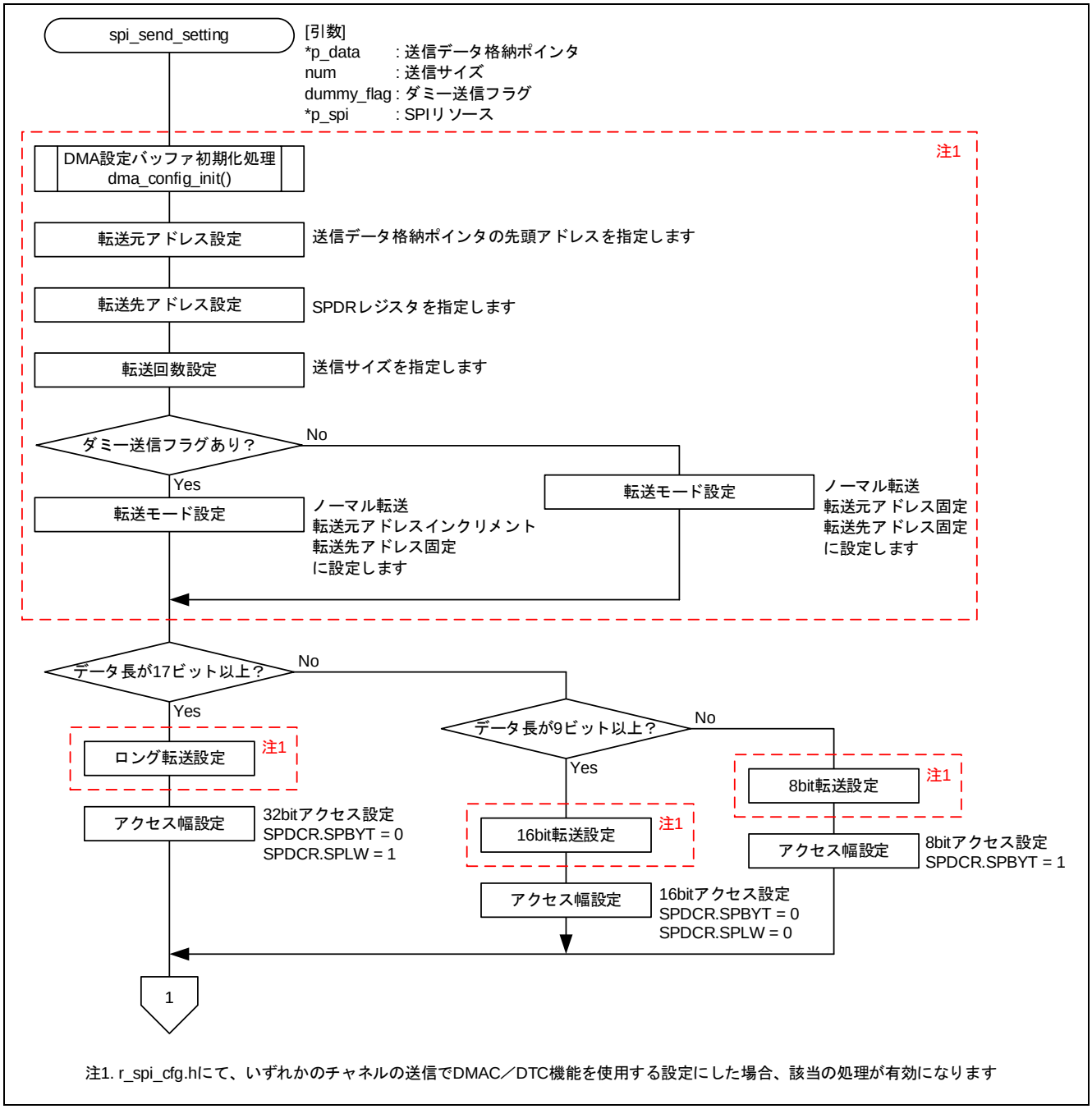
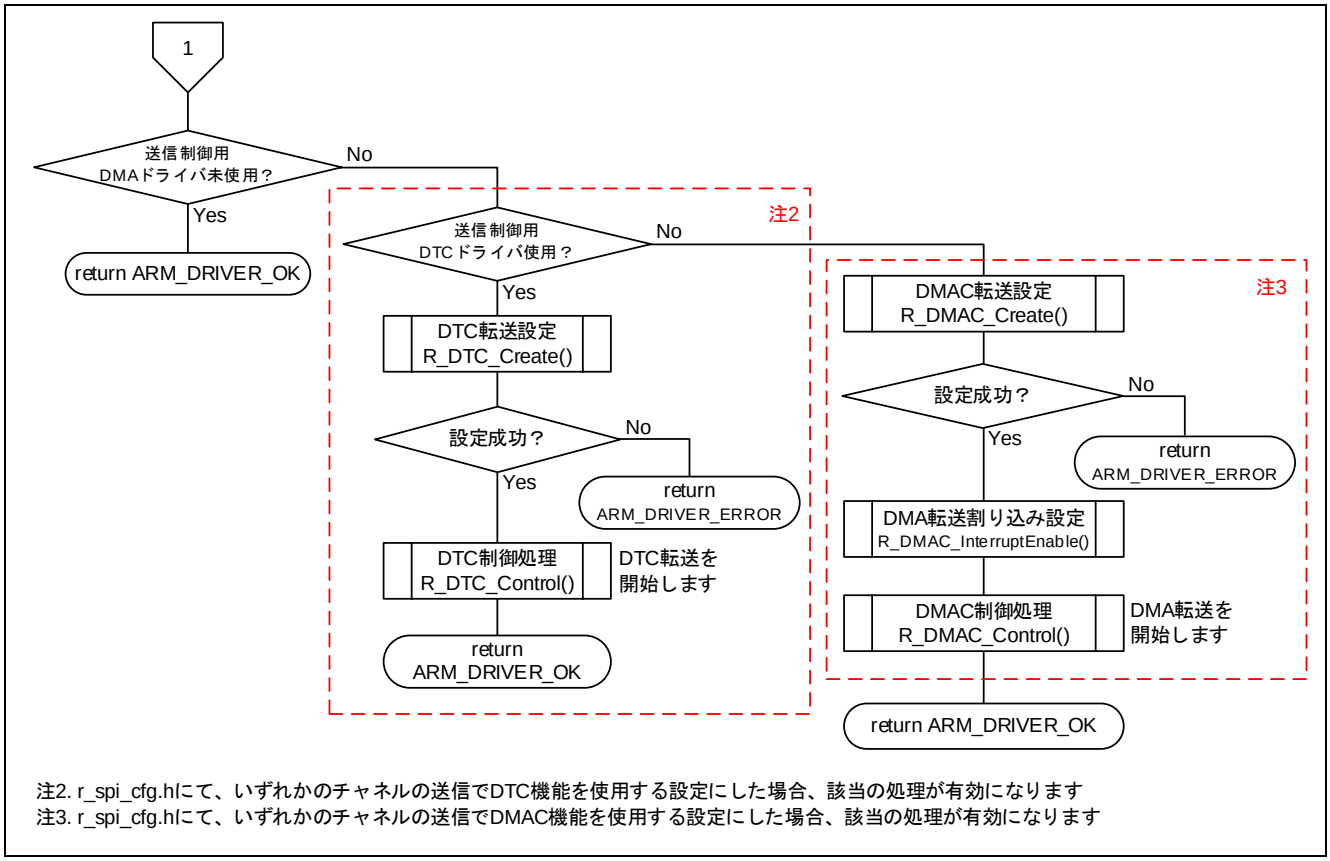


図 4-31 spi_send_setting 関数処理フロー(1/2)



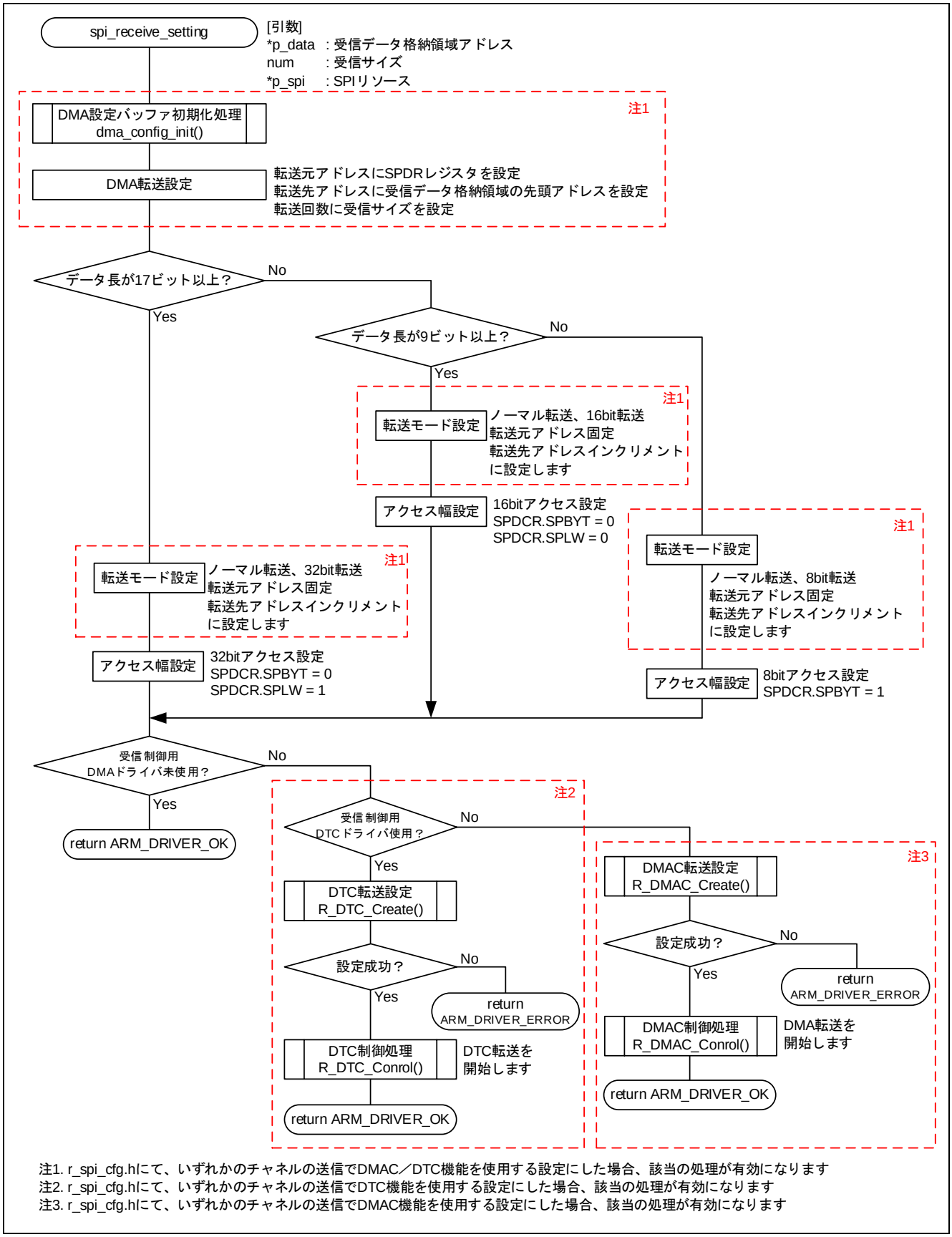
注2. r_spi_cfg.hにて、いずれかのチャネルの送信でDTC機能を使用する設定にした場合、該当の処理が有効になります
注3. r_spi_cfg.hにて、いずれかのチャネルの送信でDMAC機能を使用する設定にした場合、該当の処理が有効になります

図 4-32 spi_send_setting 関数処理フロー(2/2)

4.1.24 spi_receive_setting 関数

表 4-25 spi_receive_setting 関数仕様

書式	static int32_t spi_receive_setting(void const * const p_data, uint32_t num, st_spi_resources_t * const p_spi)
仕様説明	受信設定を行います
引数	void const * const p_data : 受信データ格納先ポインタ
	uint32_t num : 受信サイズ
	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI のリソースを指定します
戻り値	ARM_DRIVER_OK 受信設定成功
	ARM_DRIVER_ERROR 受信設定失敗 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、受信設定失敗となります
備考	-



注1. r_spi_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2. r_spi_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3. r_spi_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 4-33 spi_receive_setting 関数処理フロー

4.1.25 dma_config_init 関数

表 4-26 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	-

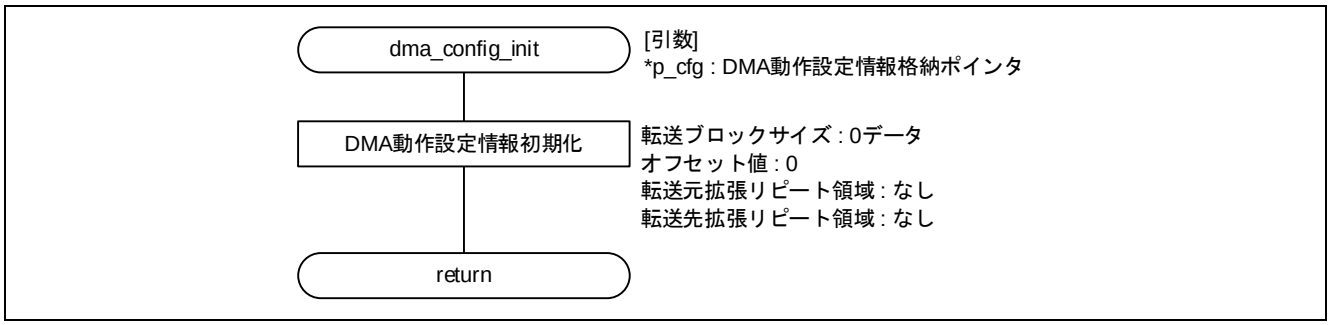


図 4-34 dma_config_init 関数処理フロー

4.1.26 r_spti_handler 関数

表 4-27 r_spti_handler 関数仕様

書式	static void r_spti_handler(st_spi_resources_t * const p_spi)
仕様説明	SPTI 割り込み処理（送信処理に割り込み使用時）
引数	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI 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 SPTI 割り込み処理です

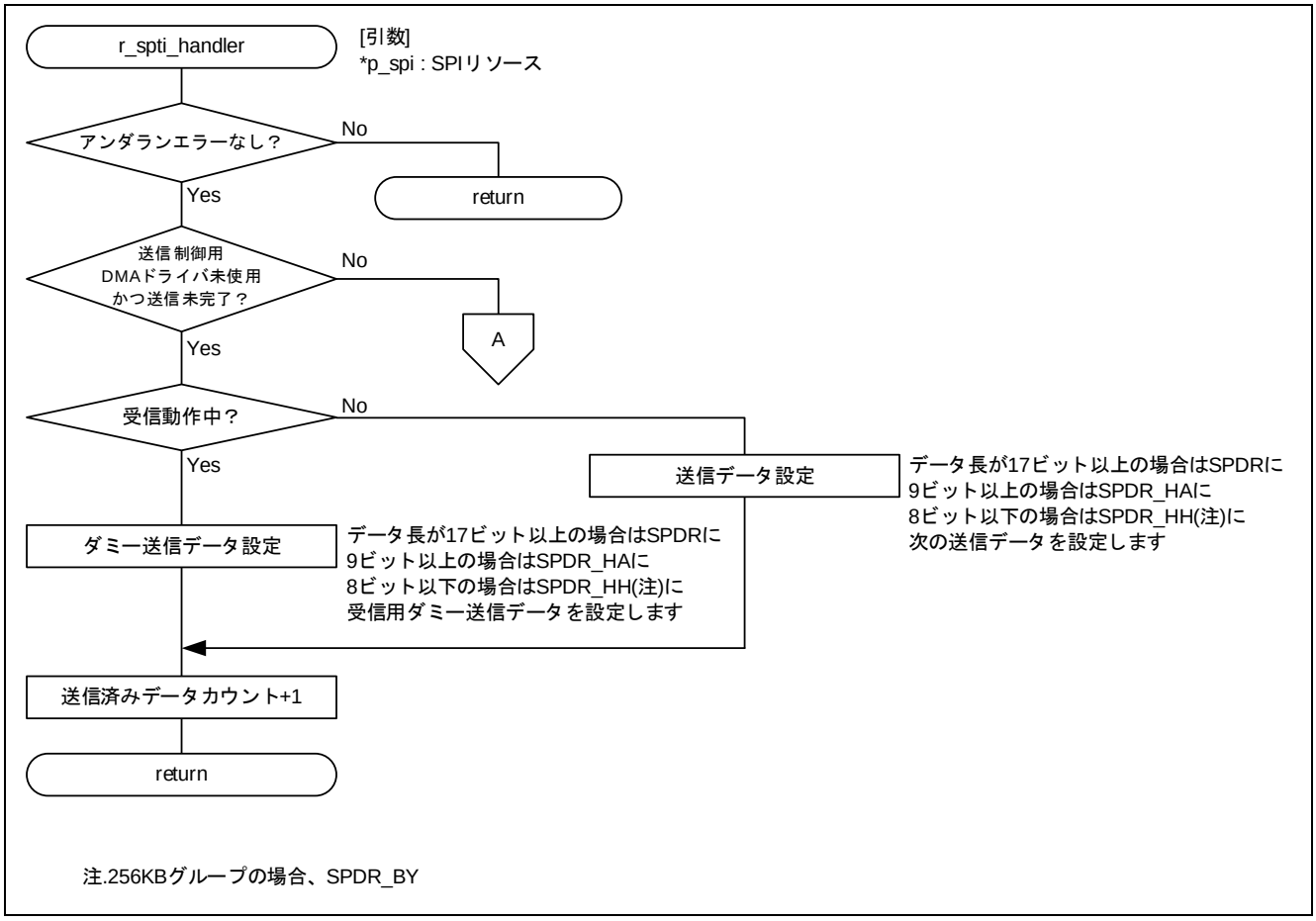
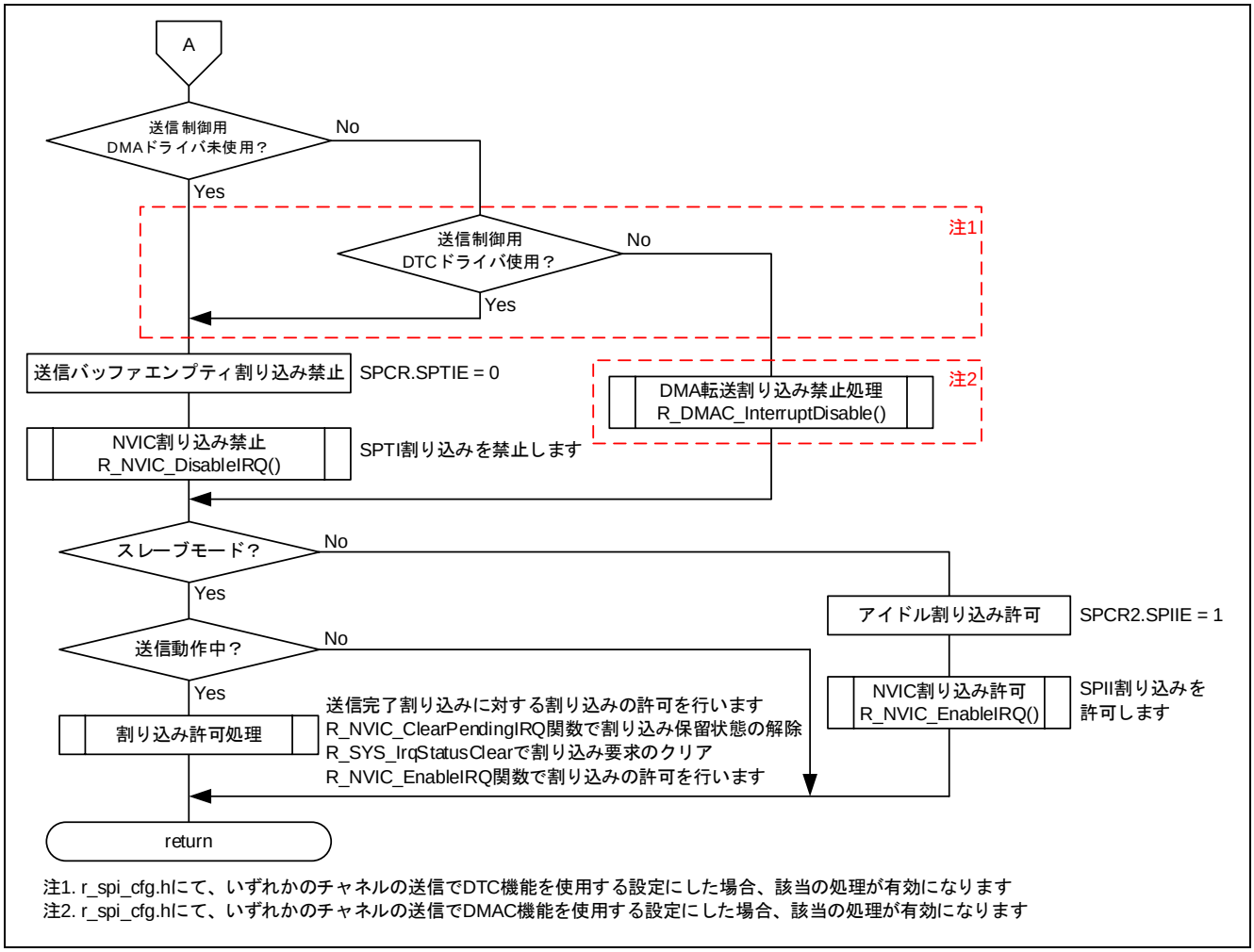


図 4-35 r_spti_handler 関数処理フロー(1/2)



注1. r_spi_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_spi_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 4-36 r_spti_handler 関数処理フロー(2/2)

4.1.27 r_spri_handler 関数

表 4-28 r_spri_handler 関数仕様

書式	static void r_spri_handler(st_spi_resources_t * const p_spi)
仕様説明	SPRI 割り込み処理（受信処理に割り込み使用時）
引数	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI 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 SPRI 割り込み処理です

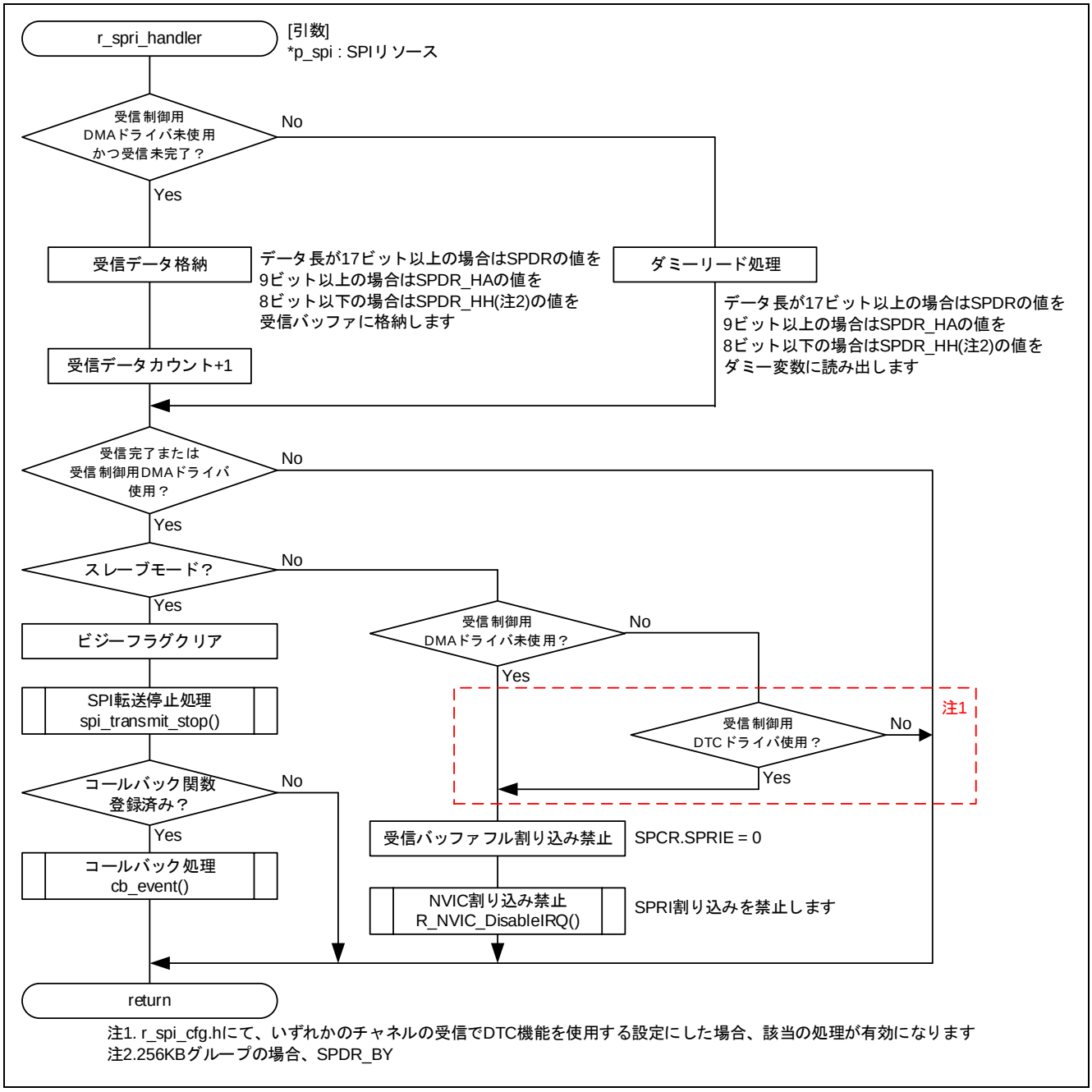


図 4-37 r_spri_handler 関数処理フロー

4.1.28 r_spii_handler 関数

表 4-29 r_spii_handler 関数仕様

書式	static void r_spii_handler(st_spi_resources_t * const p_spi)
仕様説明	SPII 割り込み処理
引数	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI のリソースを指定します
戻り値	なし
備考	-

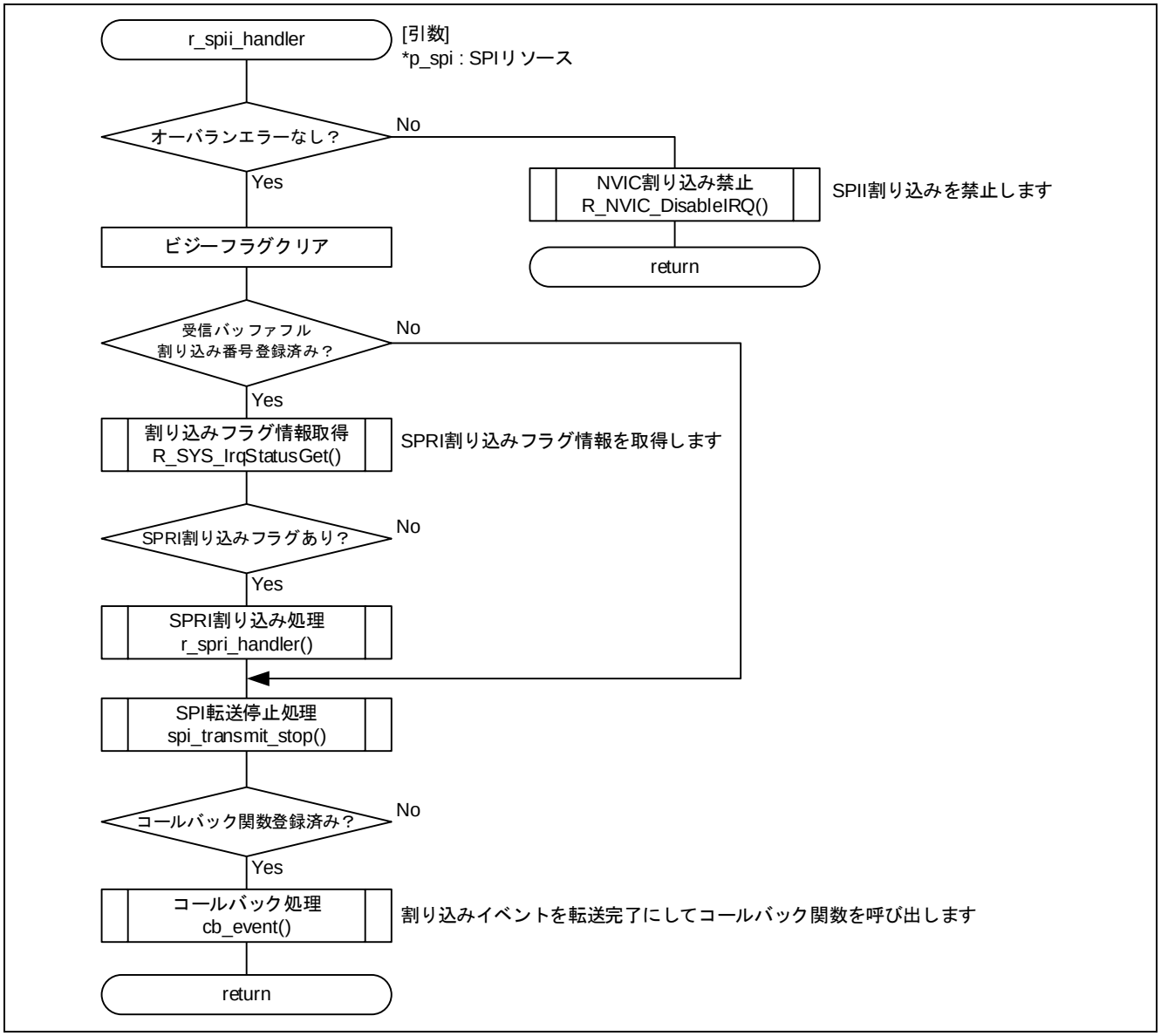


図 4-38 r_spii_handler 関数処理フロー

4.1.29 r_spei_handler 関数

表 4-30 r_spei_handler 関数仕様

書式	static void r_spei_handler(st_spi_resources_t * const p_spi)
仕様説明	SPEI 割り込み処理
引数	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI のリソースを指定します
戻り値	なし
備考	-

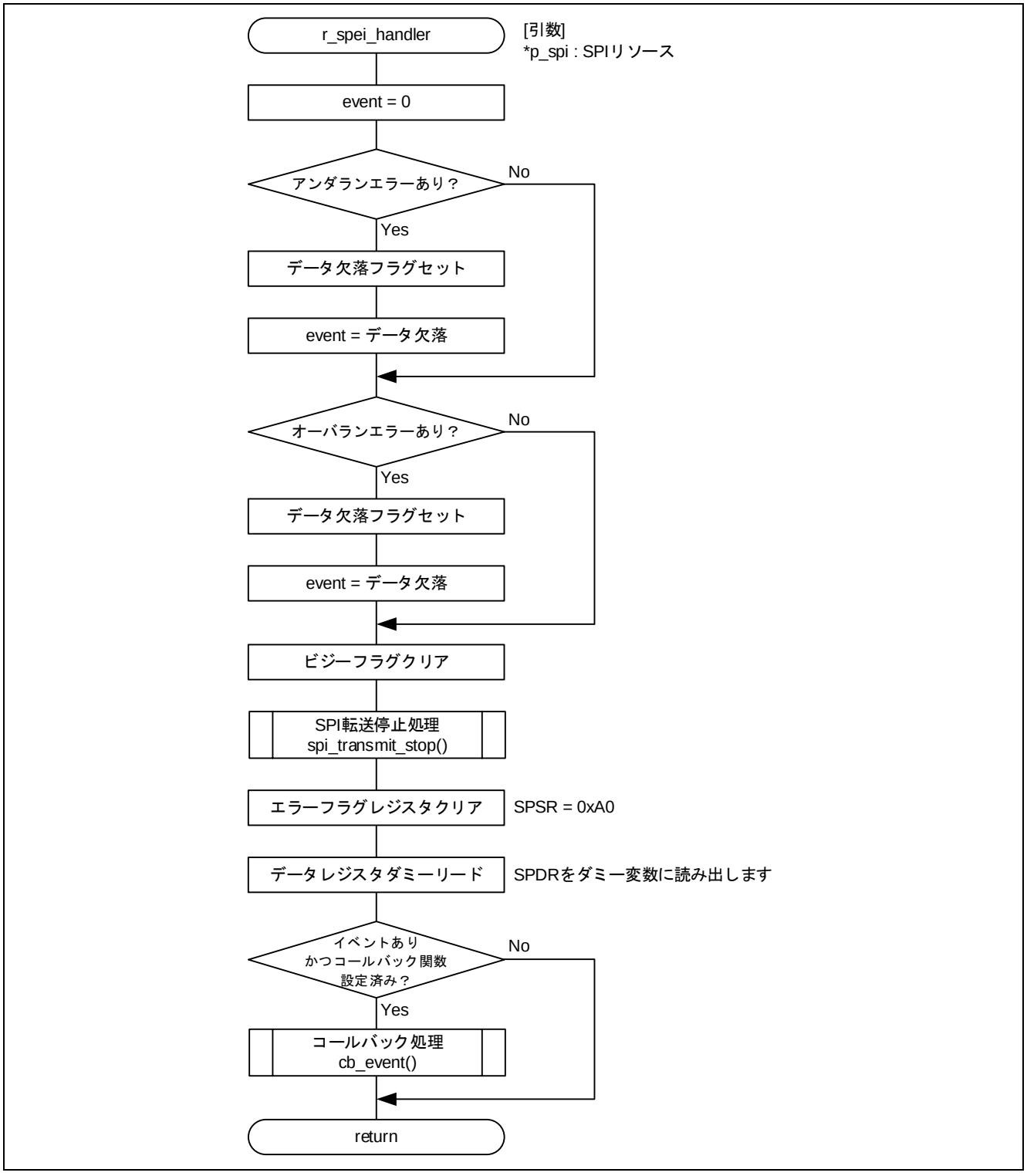


図 4-39 r_spei_handler 関数処理フロー

4.1.30 r_sptend_handler 関数

表 4-31 r_sptend_handler 関数仕様

書式	static void r_sptend_handler(st_spi_resources_t * const p_spi)
仕様説明	SPTEND 割り込み処理
引数	st_spi_resources_t * const p_spi : SPI のリソース 制御対象の SPI のリソースを指定します
戻り値	なし
備考	-

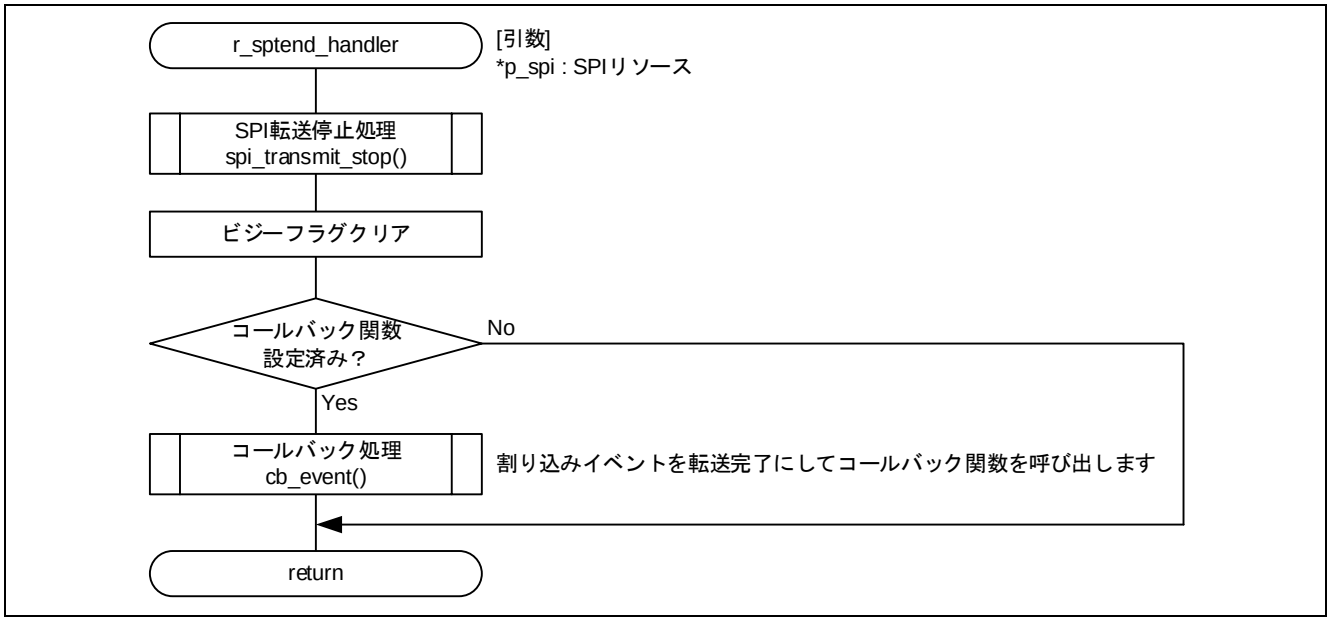


図 4-40 r_sptend_handler 関数処理フロー

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 4-32 マクロ定義一覧(1/2)

定義	値	内容
R_SPI0_ENABLE	(1)	SPI0 リソース有効定義
R_SPI1_ENABLE	(1)	SPI1 リソース有効定義
SPI_FLAG_INITIALIZED	(1U << 0)	SPI 初期化済みフラグ定義
SPI_FLAG_POWERED	(1U << 1)	モジュール解除済みフラグ定義
SPI_FLAG_CONFIGURED	(1U << 2)	モード設定済みフラグ定義
SPI_FLAG_MASTER_SEND_AVAILABLE	(1U << 3)	マスタ送信許可状態フラグ定義
SPI_FLAG_MASTER_RECEIVE_AVAILABLE	(1U << 4)	マスタ受信許可状態フラグ定義
SPI_FLAG_SLAVE_SEND_AVAILABLE	(1U << 5)	スレーブ送信許可状態フラグ定義
SPI_FLAG_SLAVE_RECEIVE_AVAILABLE	(1U << 6)	スレーブ受信許可状態フラグ定義
SPI_SPTI0_DMAC_SOURCE_ID	(0x9A)	SPTI0 用 DELS ビット設定値
SPI_SPRI0_DMAC_SOURCE_ID	(0x99)	SPRI0 用 DELS ビット設定値
SPI_SPTI1_DMAC_SOURCE_ID	(0x9F)	SPTI1 用 DELS ビット設定値
SPI_SPRI1_DMAC_SOURCE_ID	(0x9E)	SPRI1 用 DELS ビット設定値
SPI_PRV_USED_DMAC_DTC_DRV	SPI_PRV_USED_TX_DMAC_DTC_DRV | SPI_PRV_USED_RX_DMAC_DTC_DRV	DMAC/DTC ドライバ使用判定定義
SPI_PRV_USED_TX_DMAC_DTC_DRV	SPI0_TRANSMIT_CONTROL | SPI1_TRANSMIT_CONTROL	DMAC/DTC 送信処理判定定義
SPI_PRV_USED_RX_DMAC_DTC_DRV	SPI0_RECEIVE_CONTROL | SPI1_RECEIVE_CONTROL	DMAC/DTC 受信処理判定定義
SPI_PRV_USED_DMAC_DRV	SPI_PRV_USED_TX_DMAC_DRV | SPI_PRV_USED_RX_DMAC_DRV	DMAC ドライバ使用判定定義
SPI_PRV_USED_TX_DMAC_DRV	SPI_PRV_USED_TX_DMAC_DTC_DRV & 0x00FF	DMAC 送信処理判定定義
SPI_PRV_USED_RX_DMAC_DRV	SPI_PRV_USED_RX_DMAC_DTC_DRV & 0x00FF	DMAC 受信処理判定定義
SPI_PRV_USED_DTC_DRV	SPI_PRV_USED_TX_DTC_DRV | SPI_PRV_USED_RX_DTC_DRV	DTC ドライバ使用判定定義
SPI_PRV_USED_TX_DTC_DRV	SPI_PRV_USED_TX_DMAC_DTC_DRV & SPI_USED_DTC	DTC 送信処理判定定義
SPI_PRV_USED_RX_DTC_DRV	(SPI_PRV_USED_RX_DMAC_DTC_DRV & SPI_USED_DTC	DTC 受信処理判定定義

表 4-33 マクロ定義一覧(2/2)

定義	値	内容
SPI_PRV_SPCMD0_SPB_OFFSET	(8)	SPCMD0.SPB 設定用オフセット値
SPI_PRV_SPCMD0_SPB_CLR_MASK	(0xF0FF)	SPCMD0.SPB クリア用マスク値
SPI_PRV_SPCMD0_SPB_20BIT	(0x0000)	データビット長(20bit)設定値
SPI_PRV_SPCMD0_SPB_24BIT	(0x0100)	データビット長(24bit)設定値
SPI_PRV_SPCMD0_SPB_32BIT	(0x0200)	データビット長(32bit)設定値
SPI_PRV_SPCMD0_SPB_8BIT	(0x0400)	データビット長(8bit)設定値
SPI_PRV_EXEC_SEND	(0x00)	送信動作定義
SPI_PRV_EXEC_RECEIVE	(0x01)	受信動作定義
SPI_PRV_EXEC_TRANSFER	(0x02)	送受信動作定義
SPI_PRV_MASK_BRDV	(0xFFF3)	SPCMD0.BRDV 設定用マスク
SPI_PRV_BASE_BIT_MASK	(0xFFFFFFFE)	受信情報格納用データビット長ベースマスク

4.3 構造体定義

4.3.1 st_spi_resources_t 構造体

SPI のリソースを構成する構造体です。

表 4-34 st_spi_resources_t 構造体

要素名	型	内容
*reg	volatile SPI0_Type	対象の SPI 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
*ss_pin	volatile uint16_t	ソフトウェア制御による SSL0 端子(ポートレジスタ)
ss_pin_pos	uint8_t	ソフトウェア制御による SSL0 端子(端子番号)
*info	st_spi_info_t	SPI 状態情報
*xfer	st_spi_transfer_info_t	SPI 通信情報
lock_id	e_system_mcu_lock_t	SPI ロック ID
mstp_id	e_lpm_mstp_t	SPI モジュールストップ ID
spti_irq	IRQn_Type	SPTI 割り込みの NVIC 割り当て番号
spri_irq	IRQn_Type	SPRI 割り込みの NVIC 割り当て番号
spii_irq	IRQn_Type	SPII 割り込みの NVIC 割り当て番号
spei_irq	IRQn_Type	SPEI 割り込みの NVIC 割り当て番号
sptend_irq	IRQn_Type	SPTEND 割り込みの NVIC 割り当て番号
spti_iesr_val	uint32_t	SPTI 割り込みの IESR レジスタ設定値
spri_iesr_val	uint32_t	SPRI 割り込みの IESR レジスタ設定値
spii_iesr_val	uint32_t	SPII 割り込みの IESR レジスタ設定値
spei_iesr_val	uint32_t	SPEI 割り込みの IESR レジスタ設定値
sptend_iesr_val	uint32_t	SPTEND 割り込みの IESR レジスタ設定値
spti_priority	uint32_t	SPTI 割り込み優先レベル
spri_priority	uint32_t	SPRI 割り込み優先レベル
spii_priority	uint32_t	SPII 割り込み優先レベル
spei_priority	uint32_t	SPEI 割り込み優先レベル
sptend_priority	uint32_t	SPTEND 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	SPTI 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	SPRI 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
spti_callback	system_int_cb_t	SPTI コールバック関数
spri_callback	system_int_cb_t	SPRI コールバック関数
spii_callback	system_int_cb_t	SPII コールバック関数
spei_callback	system_int_cb_t	SPEI コールバック関数
sptend_callback	system_int_cb_t	SPTEND コールバック関数

4.3.2 st_spi_transfer_info_t 構造体

SPI の送受信情報を管理するための構造体です。

表 4-35 st_spi_transfer_info_t 構造体

要素名	型	内容
num	uint32_t	送受信サイズ
*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	ダミー送信データ
data_bits	uint8_t	データビット長
exec_state	uint8_t	送信/受信/送受信状態

4.3.3 st_spi_info_t 構造体

SPI の情報を管理するための構造体です。

表 4-36 st_spi_info_t 構造体

要素名	型	内容
cb_event	ARM_SPI_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
status	ARM_SPI_STATUS	SPI 通信状態
tx_status	st_spi_transfer_info_t	SPI 送受信情報
mode	uint32_t	動作モード ARM_SPI_MODE_INACTIVE：SPI 非アクティブ ARM_SPI_MODE_MASTER：マスタモード動作 ARM_SPI_MODE_SLAVE：スレーブモード動作
bps	uint32_t	ボーレート設定
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： 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) b2 : SPI モード設定済み状態(0:未設定、1:設定済み) b3： マスタ送信可否状態 (0:マスタ送信不可、1:マスタ送信可) b4： マスタ受信可否状態 (0:マスタ受信不可、1:マスタ受信可) b5： スレーブ送信可否状態 (0:スレーブ送信不可、1:スレーブ送信可) b6： スレーブ受信可否状態 (0:スレーブ受信不可、1:スレーブ受信可)

4.3.4 st_spi_reg_buf_t 構造体

レジスタ設定用バッファの構造体です。

表 4-37 st_spi_reg_buf_t 構造体

要素名	型	内容
mode	int32_t	SPI 動作モード設定バッファ
spcmd0	uint16_t	SPCMD0 レジスタ設定バッファ
spcr	uint8_t	SPCR レジスタ設定バッファ
spbr	uint8_t	SPBR レジスタ設定バッファ
data_bits	uint8_t	データビット長設定バッファ
bps	uint32_t	ボーレート設定バッファ

4.4 データテーブル定義

SPI 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 ビットレート分周設定用データテーブル

ビットレート分周設定用データテーブルは uint16_t 型で定義された SPCMD0.BRDV 設定用テーブルです。

表 4-38 ビットレート分周設定用データテーブル(gs_spi_brdv_tbl)

分周比	テーブル設定値	BRDV[1:0]		内容
		b3	b2	
0	0x0000	0	0	ベースのビットレート(注)
2	0x0004	0	1	ベースのビットレート(注)の 2 分周
4	0x0008	1	0	ベースのビットレート(注)の 4 分周
8	0x000C	1	1	ベースのビットレート(注)の 8 分周

注　ベースのビットレートは SPBR レジスタの値で決定します。SPBR 設定値は、ボーレート設定時に自動で算出されます。

4.5 外部関数の呼び出し

SPI ドライバ API から呼び出される外部関数を示します。

表 4-39 SPI ドライバ API から呼び出す外部関数と呼び出し条件(1/2)

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_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_RSPI_Pinclr_CHn(n = 0,1)	なし
	R_RSPI_Pinset_CHn(n = 0,1)	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_RSPI_Pinclr_CHn(n = 0,1)	
	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_SYS_IrqStatusClear	なし
	R_NVIC_DisableIRQ	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 4-40 SPI ドライバ API から呼び出す外部関数と呼び出し条件(2/2)

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_NVIC_ClearPendingIRQ	なし
	R_SYS_IrqStatusClear	なし
	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/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_NVIC_ClearPendingIRQ	なし
	R_SYS_IrqStatusClear	なし
	R_DTC_Create	送信/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	
GetDataCount	R_DMAC_GetTransferByte	送信/受信処理に DMAC ドライバを使用した場合
	R_DTC_GetTransferByte	送信/受信処理に DTC ドライバを使用した場合
Control	R_SYS_SystemClockFreqGet	以下のいずれかのコマンドを実行した場合 ・ARM_SPI_MODE_MASTER ・ARM_SPI_SET_BUS_SPEED ・ARM_SPI_GET_BUS_SPEED
	R_NVIC_ClearPendingIRQ	ARM_SPI_ABORT_TRANSFER コマンドを実行した場合
	R_SYS_IrqStatusClear	
	R_DMAC_Control	
	R_DMAC_InterruptDisable	
	R_NVIC_DisableIRQ	
	R_NVIC_ClearPendingIRQ	
	R_SYS_IrqStatusClear	
	R_RSPI_Pinset_CHn(n = 0,1)	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_SPI_MODE_MASTER ・ARM_SPI_MODE_SLAVE
GetStatus	-	-
GetVersion	-	-
GetCapabilities	-	-

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。

5. 使用上の注意

5.1 NVIC への SPI 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」を参照してください。

5.2 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_RSPI_Pinset_CHn(n=0,1)関数で設定、R_RSPI_Pinclr_CHn 関数で解放されます。R_RSPI_Pinset_CHn 関数は Control 関数でのマスタおよびスレーブモードの初期化、SPI 通信非アクティブ設定時に呼び出されます。R_RSPI_Pinclr_CHn 関数は PowerControl 関数でのパワーオフ設定時、または Uninitialize 関数で呼び出されます。

使用する端子は、pin.c の R_RSPI_Pinset_CHn、R_RSPI_Pinclr_CHn (n = 0,1)関数を編集して選択してください。SPI 機能で使用される各端子名には_A、_B、_C および_D という接尾語が付加されています。SPI 機能を割り当てる場合、同じ接尾語の機能端子を選択してください。(注)

端子設定変更例を図 5-1～図 5-3 に示します。

注　接尾語が同じ信号は、タイミング調整されているグループを表しています。違うグループの信号を同時に使用することはできません。例外として、SPI の「RSPCKA_C」と「MOSIA_C」は「_B」のグループと同時に使用します。また、「SSLB0_D」は「_B」のグループと同時に使用します。

```
/*****************************************************************************//**
* @brief This function sets Pin of RSPI0.
* @note  Several pin names have added _A, _B, and _C suffixes.@n
*        When assigning the SPI functions, select the functional pins with the same suffix.@n
*        Comment out the terminal of unused suffix.@n
*        When using “RSPCKA_C, MOSIA_C” added by the SPI, select the pair of RSPCKA_B and RSPCKA_C
*        and the pair of MOSIA_B and MOSIA_C. When using “SSLB0_D” added by SPI,
*        select the pair of SSLB0_D and SSLB0_B.
*****************************************************************************/
/* Function Name : R_RSPI_Pinset_CH0 */
void R_RSPI_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);

//    /* MISOA_A : P105 */
//    PFS->P105PFS_b.PMR  = 0U;
//    PFS->P105PFS_b.ASEL = 0U;
//    PFS->P105PFS_b.ISEL = 0U;
//    PFS->P105PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
//    PFS->P105PFS_b.PMR  = 1U;

/* P500 を MISOA 用端子に設定 */
    /* MISOA_B : P500 */
    PFS->P500PFS_b.ASEL = 0U;
    PFS->P500PFS_b.ISEL = 0U;
    PFS->P500PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
    PFS->P500PFS_b.PMR  = 1U;

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

/* P010 を MOSIA 用端子に設定 */
    /* MOSIA_B : P010 */
    PFS->P010PFS_b.ASEL = 0U;
    PFS->P010PFS_b.ISEL = 0U;
    PFS->P010PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
    PFS->P010PFS_b.PMR  = 1U;

    /* MOSIA_C : P501 */
//    PFS->P501PFS_b.ASEL = 0U;
//    PFS->P501PFS_b.ISEL = 0U;
//    PFS->P501PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
//    PFS->P501PFS_b.PMR  = 1U;

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

/* P011 を RSPCKA 用端子に設定 */
    /* RSPCKA_B : P011 */
    PFS->P011PFS_b.ASEL = 0U;
    PFS->P011PFS_b.ISEL = 0U;
    PFS->P011PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
    PFS->P011PFS_b.PMR  = 1U;

    /* RSPCKA_C : P502 */
//    PFS->P502PFS_b.ASEL = 0U;
//    PFS->P502PFS_b.ISEL = 0U;
//    PFS->P502PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
//    PFS->P502PFS_b.PMR  = 1U;
```

図 5-1 端子設定例(1/3)

```
//    /* SSLA0_A : P103 */
//    PFS->P103PFS_b.ASEL = 0U;
//    PFS->P103PFS_b.ISEL = 0U;
//    PFS->P103PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
//    PFS->P103PFS_b.PMR  = 1U;

/* P012 を SSLA0 用端子に設定 */
    /* SSLA0_B : P012 */
    PFS->P012PFS_b.ASEL = 0U;
    PFS->P012PFS_b.ISEL = 0U;
    PFS->P012PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
    PFS->P012PFS_b.PMR  = 1U;

//    /* SSLA1_A : P102 */
//    PFS->P102PFS_b.ASEL = 0U;
//    PFS->P102PFS_b.ISEL = 0U;
//    PFS->P102PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
//    PFS->P102PFS_b.PMR  = 1U;//

    /* SSLA1_B : P013 */
//    PFS->P013PFS_b.ASEL = 0U;
//    PFS->P013PFS_b.ISEL = 0U;
//    PFS->P013PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
//    PFS->P013PFS_b.PMR  = 1U;

//    /* SSLA2_A : P101 */
//    PFS->P101PFS_b.ASEL = 0U;
//    PFS->P101PFS_b.ISEL = 0U;
//    PFS->P101PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
//    PFS->P101PFS_b.PMR  = 1U;

    /* SSLA2_B : P014 */
//    PFS->P014PFS_b.ASEL = 0U;
//    PFS->P014PFS_b.ISEL = 0U;
//    PFS->P014PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
//    PFS->P014PFS_b.PMR  = 1U;

//    /* SSLA3_A : P100 */
//    PFS->P100PFS_b.ASEL = 0U;
//    PFS->P100PFS_b.ISEL = 0U;
//    PFS->P100PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
//    PFS->P100PFS_b.PMR  = 1U;

    /* SSLA3_B : P015 */
//    PFS->P015PFS_b.ASEL = 0U;
//    PFS->P015PFS_b.ISEL = 0U;
//    PFS->P015PFS_b.PSEL = R_PIN_PRV_RSPI_PSEL;
//    PFS->P015PFS_b.PMR  = 1U;

    /* Enable protection for PFS function (Set to PWPR register) */
    R_SYS_RegisterProtectEnable(SYSTEM_REG_PROTECT_MPC);
}/* End of function R_RSPI_Pinset_CH0() */
```

図 5-2 端子設定例(2/3)

```
/*****************************************************************************//**
* @brief This function clears the pin setting of RSPI0.
*****************************************************************************/
/* Function Name : R_RSPI_Pinclr_CH0 */
void R_RSPI_Pinclr_CH0(void)  // @suppress("API function naming")
{
    /* Disable protection for PFS function (Set to PWPR register) */
    R_SYS_RegisterProtectDisable(SYSTEM_REG_PROTECT_MPC);

//    /* MISOA_A : P105 */
//    PFS->P105PFS &= R_PIN_PRV_CLR_MASK;

/* MISOA 端子を解放 */
    /* MISOA_B : P500 */
    PFS->P500PFS &= R_PIN_PRV_CLR_MASK;

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

/* MOSIA 端子を解放 */
    /* MOSIA_B : P010 */
    PFS->P010PFS &= R_PIN_PRV_CLR_MASK;

    /* MOSIA_C : P501 */
//    PFS->P501PFS &= R_PIN_PRV_CLR_MASK;

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

/* RSPCKA 端子を解放 */
    /* RSPCKA_B : P011 */
    PFS->P011PFS &= R_PIN_PRV_CLR_MASK;

    /* RSPCKA : P502 */
//    PFS->P502PFS &= R_PIN_PRV_CLR_MASK;

//    /* SSLA0_A : P103 */
//    PFS->P103PFS &= R_PIN_PRV_CLR_MASK;

/* SSLA0 端子を解放 */
    /* SSLA0_B : P012 */
    PFS->P012PFS &= R_PIN_PRV_CLR_MASK;

//    /* SSLA1_A : P102 */
//    PFS->P102PFS &= R_PIN_PRV_CLR_MASK;

    /* SSLA1_B : P013 */
//    PFS->P013PFS &= R_PIN_PRV_CLR_MASK;

//    /* SSLA2_A : P101 */
//    PFS->P101PFS &= R_PIN_PRV_CLR_MASK;

    /* SSLA2_B : P014 */
//    PFS->P014PFS &= R_PIN_PRV_CLR_MASK;

//    /* SSLA3_A : P100 */
//    PFS->P100PFS &= R_PIN_PRV_CLR_MASK;

    /* SSLA3_B : P015 */
//    PFS->P015PFS &= 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_RSPI_Pinclr_CH0() */
```

図 5-3 端子設定例(3/3)

5.3 Control 関数による SSL 端子制御について

Control 関数(ARM_SPI_CONTROL_SS コマンド)を使用してソフトウェアで SSL 端子を制御する場合、r_spi_cfg.h ファイルの SPIn_SS_PORT、SPIn_SS_PIN（n = 0、1）にて SSL 端子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 SS 動作選択には、ARM_SPI_SS_MASTER_SW（マスタ動作時、ソフトウェア制御によるスレーブセレクト制御を使用）または、ARM_SPI_SS_SLAVE_SW（スレーブ動作時、ソフトウェア制御によるスレーブセレクト制御を監視）を指定してください。（ハードウェアによるスレーブセレクト制御に設定しないでください）

SPI0 にてソフトウェアによる SSL 制御端子を PORT107 に設定する場合の例を図 5-4 に示します。

```
・・・
/* When using the ARM_SPI_CONTROL_SS command, cancel the following comment and set the terminal
to use */
#define SPI0_SS_PORT (PORT1->PODR)              /* コメントアウトを削除、ポート 1 選択 */
#define SPI0_SS_PIN (7)                           /* SS 端子を P107 に選択 */
・・・
```

図 5-4 SSL 端子をソフトウェア制御で使用する場合の端子指定例

5.4 割り込み許可ビット 0 クリア処理のタイムアウトについて

spi_ir_flag_clear 関数内での SPCR.SPRIE ビットおよび SPCR.SPTIE ビット 0 クリア待ち処理タイムアウト時間は、r_system_cfg.h の SYSTEM_CFG_API_TIMEOUT_COUNT で定義されます。タイムアウト時間を変更する場合は、r_system_cfg.h の SYSTEM_CFG_API_TIMEOUT_COUNT の値を変更してください(注)。タイムアウト時間の設定例を図 5-5 に示します。

注　SYSTEM_CFG_API_TIMEOUT_COUNT は RE01 グループ CMSIS software package 内共通定義です。本ドライバ以外のレジスタ設定値変更待ち時間も変更されます。

```
/**********************************************************************************//**
 * @brief Time-out value of API until register value is changed.@n
 *********************************************************************************/
#define SYSTEM_CFG_API_TIMEOUT_COUNT              (0x10000000)
```

図 5-5 SYSTEM_CFG_API_TIMEOUT_COUNT 設定例

5.5 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。

5.6 スレーブモードかつ CPHA0 での通信再開について

スレーブモードで CPHA（クロック位相）を 0（立ち上がりエッジでデータサンプリング、立ち下がりエッジでデータ変化）に設定した場合、クロック信号 RSPCK の最後の半サイクル期間に通信を再開するとアンダランエラーやビットずれなどの不正な動作を行う可能性があります。コールバック関数の呼び出しもしくは GetStatus 関数によるレディー判定後、再度通信を開始する場合は RSPCK の半サイクル待ってから実行してください。スレーブモードかつ CPHA0 設定時の通信再開例を図 5-6 に示します。

```
static uint8_t tx_data[3] = {0x01, 0x02, 0x03};

/**********************************************************************************************************
* callback function
**********************************************************************************************************/
static void spi_callback(uint32_t event)
{
    switch( event )
    {
        case ARM_SPI_EVENT_TRANSFER_COMPLETE:
        {
            /* RSPCK の半サイクル待ち(通信速度 100kbps の場合 5us) */
            R_SYS_SoftwareDelay(5, SYSTEM_DELAY_UNITS_MICROSECONDS);
            /* 再開 */
            spi0Drv->Send(&tx_data[0], 3);
        }
        break;

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

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

図 5-6 スレーブモードかつ CPHA0 での通信再開例

5.7 複数データ通信時の SSL 信号制御について

本ドライバでは、1 データ通信ごとに SSL 信号がネゲート("H")します。複数データ通信時、すべてのデータ通信完了まで SSL 信号をアクティブレベル("L")で保持したい場合は、クロック同期式通信（3 線式）を使用し、ソフトウェアで SSL 信号を制御してください。複数データ通信時の SSL 信号ソフトウェア制御動作例を図 5-7 に示します。

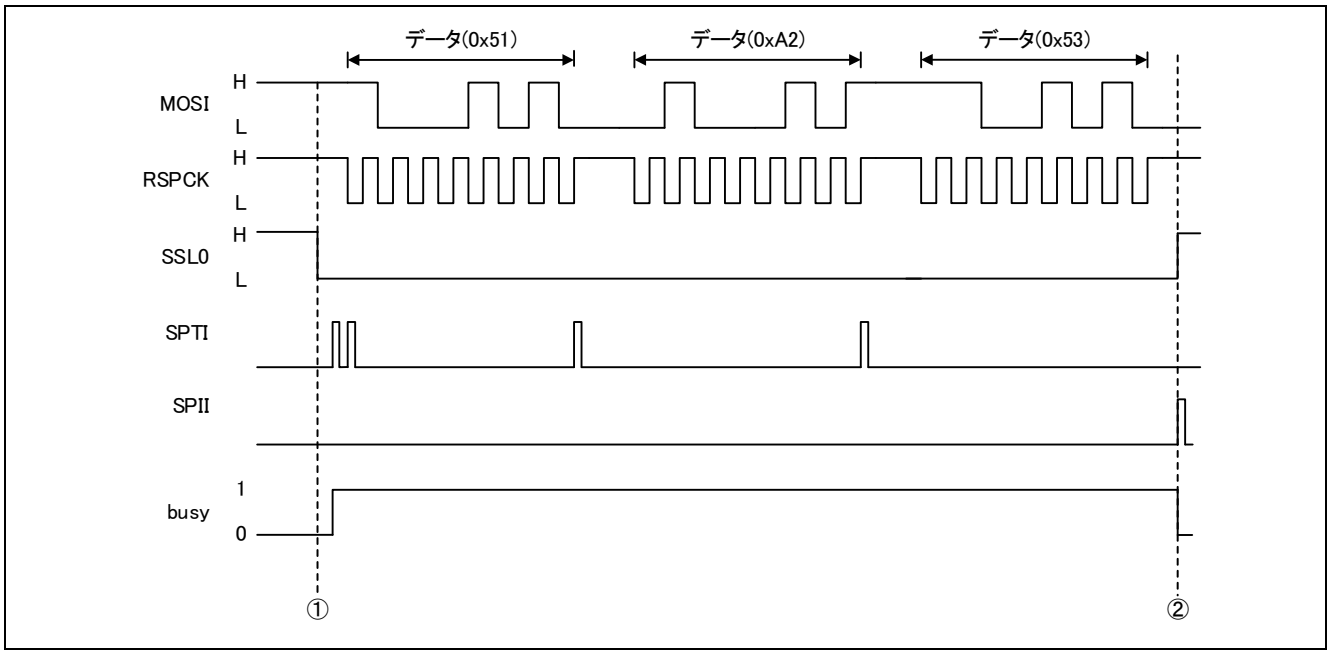


図 5-7 複数データ通信時の SSL 信号ソフトウェア制御動作例

① Control 関数の ARM_SPI_CONTROL_SS コマンド(ARM_SPI_SS_ACTIVE)にて、SSL 信号をアクティブに設定すると、SSL 信号が"L"になります。

② すべてのデータ通信終了時に、Control 関数の ARM_SPI_CONTROL_SS コマンド(ARM_SPI_SS_INACTIVE)にて、SSL 信号を非アクティブに設定すると、SSL 信号が"H"になります。

5.8 DTC 使用時の注意

本ドライバの r_spi_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_SPI_Initialize	ARM_DRIVER_ERROR 発生時
ARM_SPI_Uninitialize	ARM_SPI_Uninitialize 関数実行時
ARM_SPI_Send	ARM_DRIVER_ERROR 発生時
ARM_SPI_Receive	ARM_DRIVER_ERROR 発生時
ARM_SPI_Transfer	ARM_DRIVER_ERROR 発生時

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.27.2019	11, 119 ~ 122	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.27.2020	7~10,17,80 プログラム	SPI 制御コマンド（ビットオーダー定義）の説明を修正 SPI 制御コマンド（ビットオーダー定義）で指定するビットオーダーについて、設定が逆転していた不具合を修正
1.04	Mar.5.2020	ー プログラム (256KB)	256KB グループに対応 256KB グループの仕様を以下に示す ・8bit アクセス SPI データレジスタ名変更 (SPDR_HH → SPDR_BY)
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	Jun.10.2020	ー	誤記修正
1.07	Nov.05.2020	125 ー	"5.8 DTC 使用時の注意"を追加 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。