RE01 1500KB、256KB グループ

CMSIS ドライバ　I2C 仕様書

要旨

本書では、RE01 1500KB、256KB グループ CMSIS software package の I2C ドライバ（以下、I2C 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 4

2.　ドライバ構成 4
2.1　ファイル構成 4
2.2　ドライバ API 5
2.3　端子設定 10
2.4　NVIC 割り込み設定 11
2.5　マクロ／型定義 13
2.5.1　I2C コントロールコード定義 13
2.5.2　I2C バス速度定義 13
2.5.3　I2C アドレスフラグ定義 13
2.5.4　I2C イベントコード定義 14
2.6　構造体定義 14
2.6.1　ARM_I2C_STATUS 構造体 14
2.6.2　ARM_I2C_CAPABILITIES 構造体 15
2.7　状態遷移 16

3.　ドライバ動作説明 18
3.1　I2C の初期設定 18
3.2　マスタ送信 19
3.3　マスタ受信 21
3.4　スレーブ送信 23
3.5　スレーブ受信 25
3.6　コンフィグレーション 27
3.6.1　ノイズフィルタ定義 27
3.6.2　バス速度自動計算 有効/無効定義 27
3.6.3　SCL 立ち上がり時間/立ち下がり時間定義 27
3.6.4　バス速度設定定義 28
3.6.5　TXI 割り込み優先レベル 28
3.6.6　TEI 割り込み優先レベル 28
3.6.7　RXI 割り込み優先レベル 29
3.6.8　EEI 割り込み優先レベル 29
3.6.9　関数の RAM 配置 30

4.　ドライバ詳細情報 31
4.1　関数仕様 31
4.1.1　ARM_I2C_Initialize 関数 32
4.1.2　ARM_I2C_Uninitialize 関数 34
4.1.3　ARM_I2C_PowerControl 関数 35
4.1.4　ARM_I2C_MasterTransmit 関数 37
4.1.5　ARM_I2C_MasterReceive 関数 39
4.1.6　ARM_I2C_SlaveTransmit 関数 41
4.1.7　ARM_I2C_SlaveReceive 関数 43
4.1.8　ARM_I2C_GetDataCount 関数 45
4.1.9　ARM_I2C_Control 関数 46

4.1.10 ARM_I2C_GetStatus 関数 50
4.1.11 ARM_I2C_GetVersion 関数 51
4.1.12 ARM_I2C_GetCapabilities 関数 52
4.1.13 riic_bps_calc 関数 53
4.1.14 riic_setting 関数 55
4.1.15 riic_reset 関数 56
4.1.16 iic_txi_interrupt 関数 57
4.1.17 iic_tei_interrupt 関数 59
4.1.18 iic_rxi_interrupt 関数 60
4.1.19 iic_eei_interrupt 関数 63
4.2 マクロ／型定義 66
4.2.1 マクロ定義一覧 66
4.2.2 e_i2c_driver_state_t 定義 66
4.2.3 e_i2c_nack_state_t 定義 67
4.3 構造体定義 68
4.3.1 st_i2c_resources_t 構造体 68
4.3.2 st_i2c_reg_buf_t 構造体 69
4.3.3 st_i2c_info_t 構造体 69
4.3.4 st_i2c_transfer_info_t 構造体 70
4.4 外部関数の呼び出し 71

5. 使用上の注意 73
5.1 NVIC への I2C 割り込み登録 73
5.2 電源オープン制御レジスタ(VOCR)設定について 74
5.3 端子設定について 74
5.4 バス速度自動計算有効時の注意事項 74

6. 参考ドキュメント 76

改訂記録 77

1. 概要

I2C 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 グループおよび 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_I2C ドライバで使用する周辺機能

周辺機能	内容
I2C バスインタフェース（RIIC）	NXP 社が提唱する I2C バス(Inter-Integrated Circuit bus)インタフェース方式でのシリアル通信を実現します

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

I2C 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_I2C.h”と、ベンダ独自ファイル格納ディレクトリ内の”r_i2c_cmsis_api.c”、 ”r_i2c_cmsis_api.h”、”r_i2c_cfg.h”、”R_Drvier_I2C.h”、”pin.c”、”pin.h”の 7 個のファイルで構成されます。各ファイルの役割を表 2-1 に、ファイル構成を図 2-1 に示します。

表 2-1 R_I2C ドライバ 各ファイルの役割

ファイル名	内容
Driver_I2C.h	CMSIS Driver 標準ヘッダファイルです
R_Driver_I2C.h	CMSIS Driver の拡張ヘッダファイルです I2C ドライバを使用する場合は、本ファイルをインクルードする必要があります
r_i2c_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 I2C 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_i2c_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_i2c_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

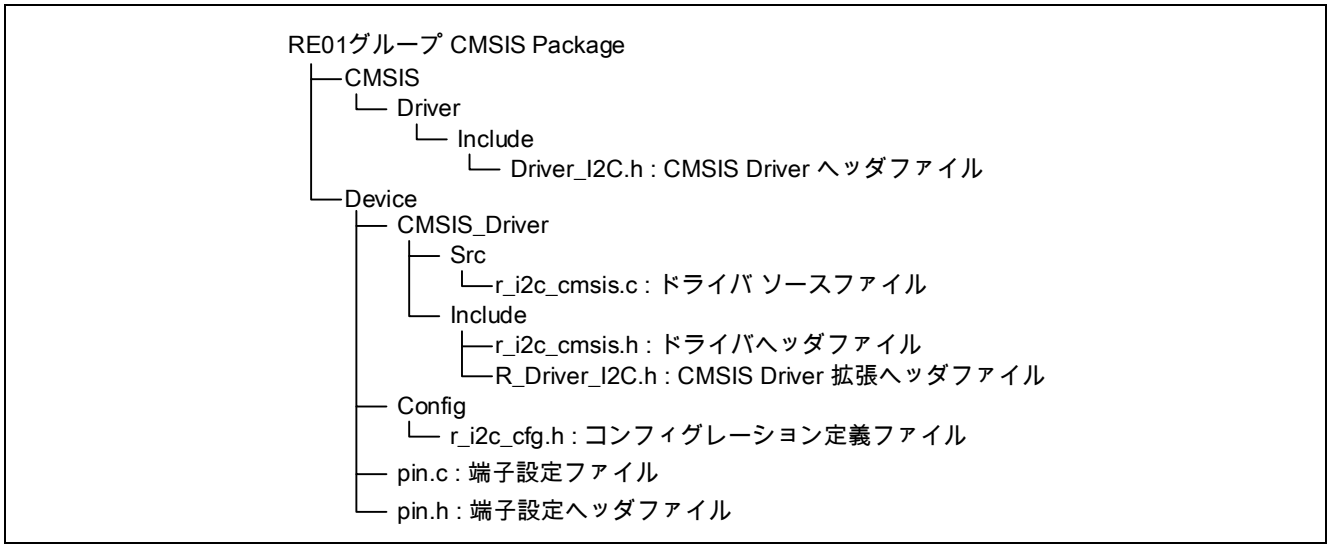

図 2-1 I2C ドライバファイル構成

2.2　ドライバ API

I2C 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。I2C 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、I2C ドライバへのアクセス例を図 2-3 ～ 図 2-6 に示します。

表 2-2 I2C ドライバのインスタンス一覧

インスタンス	内容
ARM_DRIVER_I2C Driver_I2C0	RIIC0 を使用する場合のインスタンス
ARM_DRIVER_I2C Driver_I2C1	RIIC1 を使用する場合のインスタンス

```
#include " R_Driver_I2C.h"

// I2C driver instance ( RIIC0 )
extern ARM_DRIVER_I2C Driver_I2C0;
ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
```

図 2-2 I2C ドライバ インスタンス宣言例

表 2-3 I2C ドライバ API

API	内容	参照
Initialize	I2C ドライバを初期化（RAM の初期化、端子設定、NVIC への割り込み登録）を行います	4.1.1
Uninitialize	I2C ドライバを解放（端子の解放）します モジュールストップ状態でない場合、モジュールストップ状態への遷移も行います	4.1.2
PowerControl	I2C 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	4.1.3
MasterTransmit	マスタ送信を開始します	4.1.4
MasterReceive	マスタ受信を開始します	4.1.5
SlaveTransmit	スレーブ送信を開始します	4.1.6
SlaveReceive	スレーブ受信を開始します	4.1.7
GetDataCount	送信データ数を取得します	4.1.8
Control	I2C の制御コマンドを実行します 制御コマンドについては「表 2-4 コマンド一覧」を参照	4.1.9
GetStatus	I2C の状態を取得します	4.1.10
GetVersion	I2C ドライバのバージョンを取得します	4.1.11
GetCapabilities	I2C ドライバの機能を取得します	4.1.12

表 2-4 コマンド一覧

コマンド	内容
ARM_I2C_OWN_ADDRESS	自身のスレーブアドレスを設定します 引数には 7 ビットアドレスを設定してください
ARM_I2C_BUS_SPEED	I2C バス速度を設定します 引数には以下のいずれかを設定してください ARM_I2C_BUS_SPEED_STANDARD : スタンダートスピード(100kbps) ARM_I2C_BUS_SPEED_FAST : ファストスピード(400kbps)
ARM_I2C_BUS_CLEAR	バスクリアを行います
ARM_I2C_ABORT_TRANSFER	送受信を中断します

```
#include " R_Driver_I2C.h"

static void callback(uint32_t event);

// I2C driver instance ( RIIC0 )
extern ARM_DRIVER_I2C Driver_I2C0;
ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;

// Transmit data
static uint8_t tx_data[6] = {0x00,0x11,0x12,0x13,0x14,0x15};

main()
{
    (void)i2cDev0->Initialize(callback);            /* I2C ドライバ初期化 */
    (void)i2cDev0->PowerControl(ARM_POWER_FULL);    /* I2C のモジュールストップ解除 */
    (void)i2cDev0->Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_FAST);
                                                  /* バス速度 = ファストスピード(400kbps) */
    (void)i2cDev0->MasterTransmit(0x01, tx_data, 6, false);  /* マスタ送信開始
                                                              送信先デバイスアドレス:0x01
                                                              送信バッファ : tx_data
                                                              送信サイズ : 6 バイト
                                                              ペンディングモード :false */
    while(1);
}

/**********************************************************************************************
* callback function
**********************************************************************************************/
static void callback(uint32_t event)
{
    if (ARM_I2C_EVENT_TRANSFER_DONE == event)
    {
        /* 正常に通信完了した場合の処理を記述 */
    }
    else
    {
        /* 通信異常が発生した場合の処理を記述 */
    }
}
```

図 2-3 I2C ドライバへのアクセス例（マスタ送信）

```
#include " R_Driver_I2C.h"

static void callback(uint32_t event);

// I2C driver instance ( RIIC0 )
extern ARM_DRIVER_I2C Driver_I2C0;
ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;

// Receive Buffer
static uint8_t rx_data[6];

main()
{

    (void)i2cDev0->Initialize(callback);            /* I2C ドライバ初期化 */
    (void)i2cDev0->PowerControl(ARM_POWER_FULL);    /* I2C のモジュールストップ解除 */
    (void)i2cDev0->Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_FAST);
                                                    /* バス速度 = ファストスピード(400kbps) */
    (void)i2cDev0-> MasterReceive (0x01, rx_data, 6, false);  /* マスタ受信開始
                                                      受信先デバイスアドレス:0x01
                                                      受信バッファ : rx_data
                                                      受信サイズ : 6 バイト */

    while(1);
}

/***********************************************************************************************
* callback function
***********************************************************************************************/
static void callback(uint32_t event)
{
    if (ARM_I2C_EVENT_TRANSFER_DONE == event)
    {
        /* 正常に通信完了した場合の処理を記述 */
    }
    else
    {
        /* 通信異常が発生した場合の処理を記述 */
    }
}
```

図 2-4 I2C ドライバへのアクセス例（マスタ受信）

```
#include " R_Driver_I2C.h"

static void callback(uint32_t event);

// I2C driver instance ( RIIC0 )
extern ARM_DRIVER_I2C Driver_I2C0;
ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;

// Transmit data
static uint8_t tx_data[6] = {0x00,0x11,0x12,0x13,0x14,0x15};

main()
{
    uint32_t bus_speed = ARM_I2C_BUS_SPEED_FAST;

    (void)i2cDev0->Initialize(callback);           /* I2C ドライバ初期化 */
    (void)i2cDev0->PowerControl(ARM_POWER_FULL);    /* I2C のモジュールストップ解除 */
    (void)i2cDev0-> SlaveTramsmit (tx_data, 6);     /* スレーブ送信開始（送信サイズ 6 バイト） */
    while(1);
}

/**********************************************************************************************************
* callback function
**********************************************************************************************************/
static void callback(uint32_t event)
{
    switch (event)
    {
    case ARM_I2C_EVENT_SLAVE_RECEIVE:
        /* アドレス+W 受信時の処理を記載 */
    break;

    case ARM_I2C_EVENT_SLAVE_TRANSMIT
        /* アドレス+R 受信時の処理を記載 */
    break;

    case ARM_I2C_EVENT_TRANSFER_DONE | ARM_I2C_EVENT_GENERAL_CALL:
    case ARM_I2C_EVENT_TRANSFER_DONE:
        /* 正常に通信完了した場合の処理を記述 */
    break;

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

図 2-5 I2C ドライバへのアクセス例（スレーブ受信）

```
#include " R_Driver_I2C.h"

static void callback(uint32_t event);

// I2C driver instance ( RIIC0 )
extern ARM_DRIVER_I2C Driver_I2C0;
ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;

// Receive Buffer
static uint8_t rx_data[6];

main()
{
    uint32_t bus_speed = ARM_I2C_BUS_SPEED_FAST;

    (void)i2cDev0->Initialize(callback);            /* I2C ドライバ初期化 */
    (void)i2cDev0->PowerControl(ARM_POWER_FULL);    /* I2C のモジュールストップ解除 */
    (void)i2cDev0-> SlaveReceive (rx_data, 6);       /* スレーブ受信開始（受信サイズ 6 バイト） */
    while(1);
}

/**********************************************************************************************************
* callback function
**********************************************************************************************************/
static void callback(uint32_t event)
{
    switch (event)
    {
    case ARM_I2C_EVENT_SLAVE_RECEIVE:
        /* アドレス+W 受信時の処理を記載 */
    break;

    case ARM_I2C_EVENT_SLAVE_TRANSMIT
        /* アドレス+R 受信時の処理を記載 */
    break;

    case ARM_I2C_EVENT_TRANSFER_DONE | ARM_I2C_EVENT_GENERAL_CALL:
    case ARM_I2C_EVENT_TRANSFER_DONE:
        /* 正常に通信完了した場合の処理を記述 */
    break;

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

図 2-6 I2C ドライバへのアクセス例（スレーブ受信）

2.3　端子設定

本ドライバで使用する端子は、pin.c の R_RIIC_Pinset_CHn(n=0、1)関数で設定、R_RIIC_Pinclr_CHn 関数で解除されます。R_RIIC_Pinset_CHn 関数は、Initialize 関数から、R_RIIC_Pinclr_CHn 関数は Uninitialize 関数から呼び出されます。

使用する端子は、pin.c の R_RIIC_Pinset_CHn、R_RIIC_Pinclr_CHn (n = 0,1)関数を編集して選択してください。端子設定例を図 2-6 に示します。

```
/*****************************************************************************//**
* @brief This function sets Pin of RIIC0.
*******************************************************************************/
/* Function Name : R_RIIC_Pinset_CH0 */
void R_RIIC_Pinset_CH0(void)  // @suppress("Source file naming") @suppress("API function naming")
{
    /* Disable protection for PFS function (Set to PWPR register) */
    R_SYS_RegisterProtectDisable(SYSTEM_REG_PROTECT_MPC);

    /* Set P810 as SCL0 */
    /* SCL0 : P810 */
    PFS->P810PFS_b.ASEL = 0U;  /* 0: Do not use as an analog pin, 1: Use as an analog pin. */
    PFS->P810PFS_b.ISEL = 0U;/* 0: Do not use as an IRQn input pin,  1: Use as an IRQn input pin. */
    PFS->P810PFS_b.DSCR = 0x2U;  /* When using RIIC : DSCR = 10b */
    PFS->P810PFS_b.PSEL = R_PIN_PRV_RIIC_PSEL;
    PFS->P810PFS_b.PMR  = 1U;/* 0: Use the pin as a general I/O port,
                                1: Use the pin as a peripheral module. */

    /* Set P809 as SDA0 */
    /* SDA0 : P809 */
    PFS->P809PFS_b.ASEL = 0U;  /* 0: Do not use as an analog pin, 1: Use as an analog pin. */
    PFS->P809PFS_b.ISEL = 0U;  /* 0: Do not use as an IRQn input pin,
                                  1: Use as an IRQn input pin. */
    PFS->P809PFS_b.DSCR = 0x2U;  /* When using RIIC : DSCR = 10b */
    PFS->P809PFS_b.PSEL = R_PIN_PRV_RIIC_PSEL;
    PFS->P809PFS_b.PMR  = 1U;  /* 0: Use the pin as a general I/O port,
                                  1: Use the pin as a peripheral module. */

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

}/* End of function R_RIIC_Pinset_CH0() */

/*****************************************************************************//**
* @brief This function clears the pin setting of RIIC0.
*******************************************************************************/
/* Function Name : R_RIIC_Pinclr_CH0 */
void R_RIIC_Pinclr_CH0(void)  // @suppress("Source file naming") @suppress("API function naming")
{
    /* Disable protection for PFS function (Set to PWPR register) */
    R_SYS_RegisterProtectDisable(SYSTEM_REG_PROTECT_MPC);

    /* Release SCL0 pin */
    /* SCL0 : P810 */
    PFS->P810PFS &= R_PIN_PRV_CLR_MASK;

    /* Release SDA0 pin */
    /* SDA0 : P809 */
    PFS->P809PFS &= 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_RIIC_Pinclr_CH0() */
```

図 2-7 端子設定例

2.4　NVIC 割り込み設定

本ドライバで使用する割り込みを表 2-5 に示します。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）に登録してください。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」を参照ください。NVIC への割り込み登録例（r_system_cfg.h の記載例）を図 2-8 に示します。

表 2-5 I2C ドライバで使用する NVIC 登録定義

割り込み	NVIC 登録定義(n = 0,1)
I2C 受信データフル割り込み(RXI)	SYSTEM_CFG_EVENT_NUMBER_IICn_RXI
I2C 送信完了割り込み(TEI)	SYSTEM_CFG_EVENT_NUMBER_IICn_TEI
I2C 送信データエンプティ割り込み(TXI)	SYSTEM_CFG_EVENT_NUMBER_IICn_TXI
I2C 通信エラー・イベント発生割り込み(EEI)	SYSTEM_CFG_EVENT_NUMBER_IICn_EEI

```
・・・

#define SYSTEM_CFG_EVENT_NUMBER_ADC140_WCMPM
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 0/4/8/12/16/20/24/28 only */

#define SYSTEM_CFG_EVENT_NUMBER_IIC0_RXI
    (SYSTEM_IRQ_EVENT_NUMBER0)  /*!< Numbers 0/4/8/12/16/20/24/28 only */

#define SYSTEM_CFG_EVENT_NUMBER_CCC_PRD
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 0/4/8/12/16/20/24/28 only */

・・・

#define SYSTEM_CFG_EVENT_NUMBER_ADC140_WCMPUM
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 1/5/9/13/17/21/25/29 only */

#define SYSTEM_CFG_EVENT_NUMBER_IIC0_TXI
    (SYSTEM_IRQ_EVENT_NUMBER1)  /*!< Numbers 1/5/9/13/17/21/25/29 only */

#define SYSTEM_CFG_EVENT_NUMBER_DOC_DOPCI
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 1/5/9/13/17/21/25/29 only */

・・・

#define SYSTEM_CFG_EVENT_NUMBER_ADC140_GCADI
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 2/6/10/14/18/22/26/30 only */

#define SYSTEM_CFG_EVENT_NUMBER_IIC0_TEI
    (SYSTEM_IRQ_EVENT_NUMBER2)  /*!< Numbers 2/6/10/14/18/22/26/30 only */

#define SYSTEM_CFG_EVENT_NUMBER_CAC_MENDI
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 2/6/10/14/18/22/26/30 only */

・・・

#define SYSTEM_CFG_EVENT_NUMBER_ACMP_CMPI
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 3/7/11/15/19/23/27/31 only */

#define SYSTEM_CFG_EVENT_NUMBER_IIC0_EEI
    (SYSTEM_IRQ_EVENT_NUMBER3)  /*!< Numbers 3/7/11/15/19/23/27/31 only */

#define SYSTEM_CFG_EVENT_NUMBER_CAC_OVFI
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 3/7/11/15/19/23/27/31 only */
```

図 2-8 r_system_cfg.h での NVIC への割り込み登録例（RIIC0 を使用）

2.5　マクロ／型定義

I2C ドライバで、ユーザが参照可能なマクロ／型定義を Driver_I2C.h ファイルで定義しています。

2.5.1　I2C コントロールコード定義

I2C コントロールコードは、Control 関数で使用する I2C 制御コマンドです。

表 2-6 I2C コントロールコード一覧

定義	値	内容
ARM_I2C_OWN_ADDRESS	(0x01)	自身のスレーブアドレス設定コマンド
ARM_I2C_BUS_SPEED	(0x02)	I2C バス速度設定コマンド
ARM_I2C_BUS_CLEAR	(0x03)	バスクリアコマンド
ARM_I2C_ABORT_TRANSFER	(0x04)	送受信中断コマンド

2.5.2　I2C バス速度定義

I2C バス速度設定コマンド(ARM_I2C_BUS_SPEED)で指定する速度の定義です。

表 2-7 I2C バス速度定義一覧

定義	値	内容
ARM_I2C_BUS_SPEED_STANDARD	(0x01)	スタンダードスピード(100kbps)
ARM_I2C_BUS_SPEED_FAST	(0x02)	ファストスピード(400kbps)
ARM_I2C_BUS_SPEED_FAST_PLUS	(0x03)	使用禁止(注 1)
ARM_I2C_BUS_SPEED_HIGH	(0x04)	使用禁止(注 1)

注1. 本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。

2.5.3　I2C アドレスフラグ定義

スレーブアドレス設定時のオプション定義です。

表 2-8 I2C アドレスフラグ定義一覧

定義	値	内容
ARM_I2C_ADDRESS_10BIT	(0x4000)	未使用(注 2)
ARM_I2C_ADDRESS_GC	(0x8000)	ジェネラルコール定義 スレーブアドレス設定時に自身のアドレスと OR で指定した場合、ジェネラルコールへの応答が有効になります

注2. 本ドライバではサポートしていません。

2.5.4　I2C イベントコード定義

コールバック関数で通知されるイベント定義です。複数イベントが同時に発生した場合は、OR 結合した値を通知します。

表 2-9 I2C イベントコード一覧

定義	値	内容
ARM_I2C_EVENT_TRANSFER_DONE	(1UL << 0)	通信が完了しました
ARM_I2C_EVENT_TRANSFER_INCOMPLETE	(1UL << 1)	通信が不完全終了しました
ARM_I2C_EVENT_SLAVE_TRANSMIT	(1UL << 2)	スレーブ送信を開始しました
ARM_I2C_EVENT_SLAVE_RECEIVE	(1UL << 3)	スレーブ受信を開始しました
ARM_I2C_EVENT_ADDRESS_NACK	(1UL << 4)	スレーブからアドレスに対する NACK を受信しました
ARM_I2C_EVENT_GENERAL_CALL	(1UL << 5)	ジェネラルコールを受け付けました
ARM_I2C_EVENT_ARBITRATION_LOST	(1UL << 6)	アービトレーションロストが発生しました
ARM_I2C_EVENT_BUS_ERROR	(1UL << 7)	未使用
ARM_I2C_EVENT_BUS_CLEAR	(1UL << 8)	未使用

2.6　構造体定義

I2C ドライバでは、ユーザが参照可能な構造体定義を Driver_I2C.h ファイルで定義しています。

2.6.1　ARM_I2C_STATUS 構造体

GetStatus 関数で I2C の状態を返すときに使用する構造体です。

表 2-10 ARM_I2C_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)
mode	uint32_t:1	モードを示します 0: スレーブ 1: マスタ
direction	uint32_t:1	送信/受信を示します 0: 送信 1: 受信
general_call	uint32_t:1	ジェネラルコールの受付状態を示します 0: ジェネラルコール未受信 1: ジェネラルコール受信
arbitration_lost	uint32_t:1	アービトレーションロストの検出状態を示します 0: アービトレーションロスト未検出 1: アービトレーションロスト検出
bus_error	uint32_t:1	未使用(0 固定)
reserved	uint32_t:26	予約領域

2.6.2　ARM_I2C_CAPABILITIES 構造体

GetCapabilities 関数で I2C の機能を返すときに使用する構造体です。

表 2-11 ARM_I2C_CAPABILITIES 構造体

要素名	型	内容
address_10_bit	uint32_t:1	10bit アドレスの有効/無効 0(無効)を返します
reserved	uint32_t:31	予約領域

2.7　状態遷移

I2C ドライバの状態遷移図を図 2-9 に、各状態でのイベント動作を表 2-12 に示します。

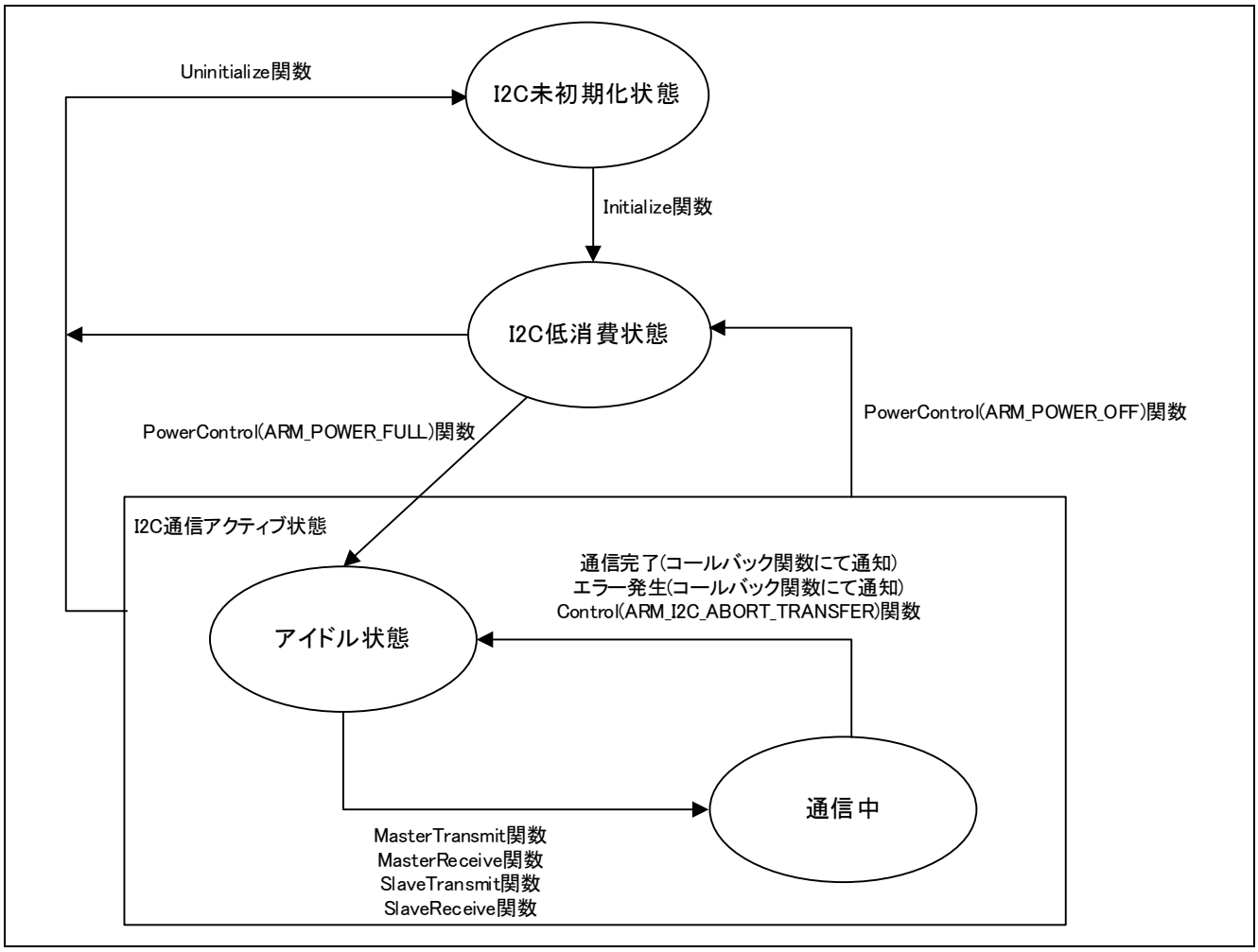

図 2-9 I2C ドライバの状態遷移

表 2-12 I2C 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
I2C 未初期化状態	リセット解除後の I2C ドライバの状態です	Initialize 関数の実行	I2C 低消費状態に遷移
I2C 低消費状態	I2C 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	I2C 未初期化状態に遷移
		PowerControl(ARM_POWER_FULL)関数の実行	I2C 通信アクティブ状態（アイドル状態）に遷移
I2C 通信アクティブ状態(アイドル状態)	通信待ち状態です	Uninitialize 関数の実行	I2C 未初期化状態に遷移
		PowerControl(ARM_POWER_OFF)関数の実行	I2C 低消費状態に遷移
		MasterTransmit 関数の実行	マスタ送信による通信中に遷移
		MasterReceive 関数の実行	マスタ受信による通信中に遷移
		SlaveTransmit 関数の実行	スレーブ送信による通信中に遷移
		SlaveReceive 関数の実行	スレーブ受信による通信中に遷移
		Control(ARM_I2C_OWN_ADDRESS)関数の実行	自身のスレーブアドレスを設定します
		Control(ARM_I2C_BUS_SPEED)関数の実行	バス速度を設定します
		Control(ARM_I2C_BUS_CLEAR)関数の実行	バスクリア処理を実行します
I2C 通信アクティブ状態（通信中）	I2C の通信実行中です	Uninitialize 関数の実行	I2C 未初期化状態に遷移
		PowerControl(ARM_POWER_OFF)関数の実行	I2C 低消費状態に遷移
		通信の完了	アイドル状態に遷移し、コールバック関数を呼び出します(注 2)
		エラー発生	アイドル状態に遷移し、コールバック関数を呼び出します(注 2)
		Control(ARM_I2C_ABORT_TRANSFER)関数の実行	通信を中断し、アイドル状態に遷移します
		Control(ARM_I2C_BUS_CLEAR)関数の実行	バスクリア処理を実行します

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

注2. 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

I2C ドライバは I2C 通信機能を実現します。本章では、I2C ドライバの呼出し手順を示します。

3.1　　I2C の初期設定

I2C の初期設定手順を図 3-1 に示します。

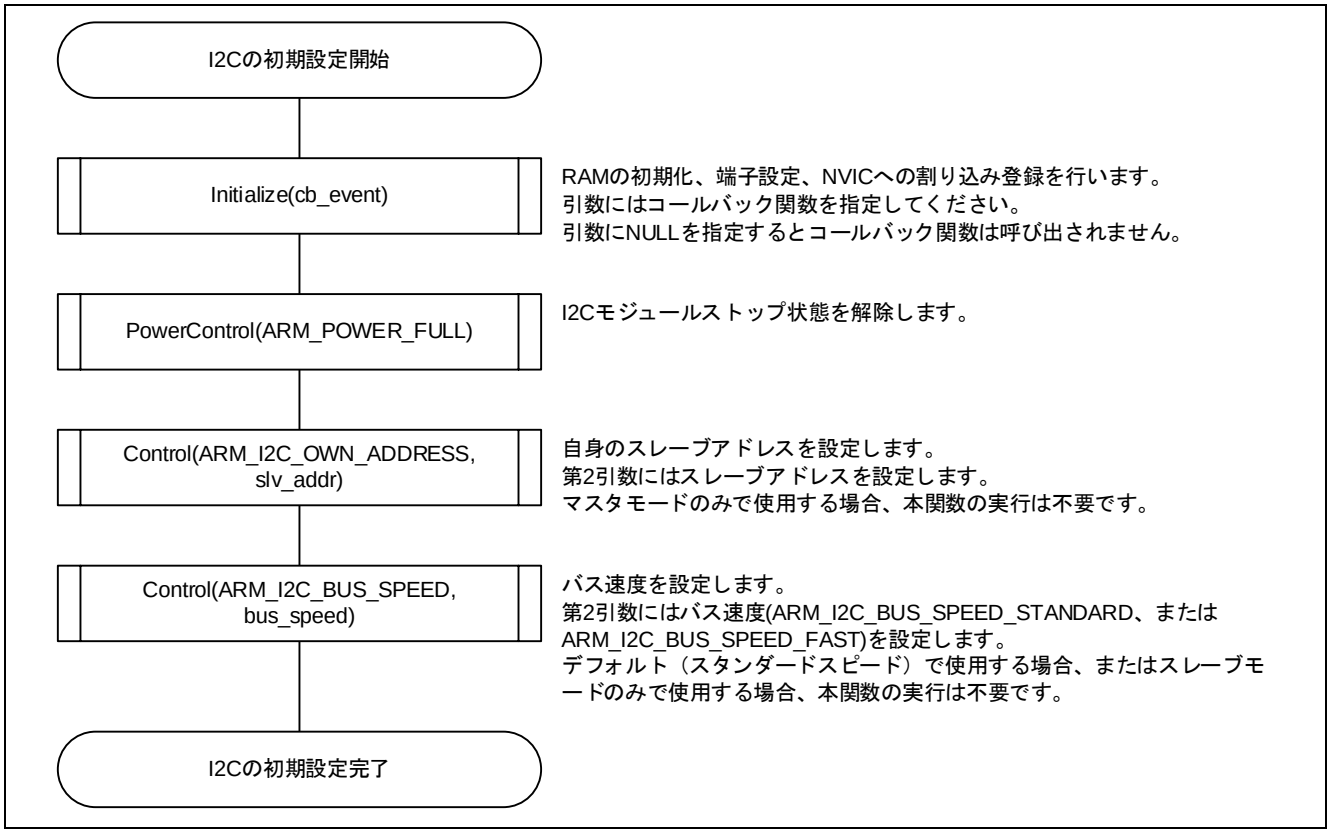

図 3-1 I2C 初期化手順

3.2　マスタ送信

マスタ送信を開始すると、スタートコンディションを出力したのち、転送先デバイスのアドレス(W)信号、送信データの順で出力します。すべてのデータが出力完了すると、ペンディング動作無しの場合、ストップコンディションを出力してマスタ送信を完了します。

ペンディング動作を有効にした場合、ストップコンディションを出力しません。再度マスタ送信、マスタ受信を実行するとリスタートコンディションからマスタ送信、またはマスタ受信を開始します。

通信デバイスのアドレスに RIIC_ADDR_NONE を指定すると、スタートコンディションとストップコンディションのみ出力します。（ペンディング動作ありに設定していた場合でもストップコンディションを出力します）

I2C でマスタ送信を行う手順を図 3-2 に示します。

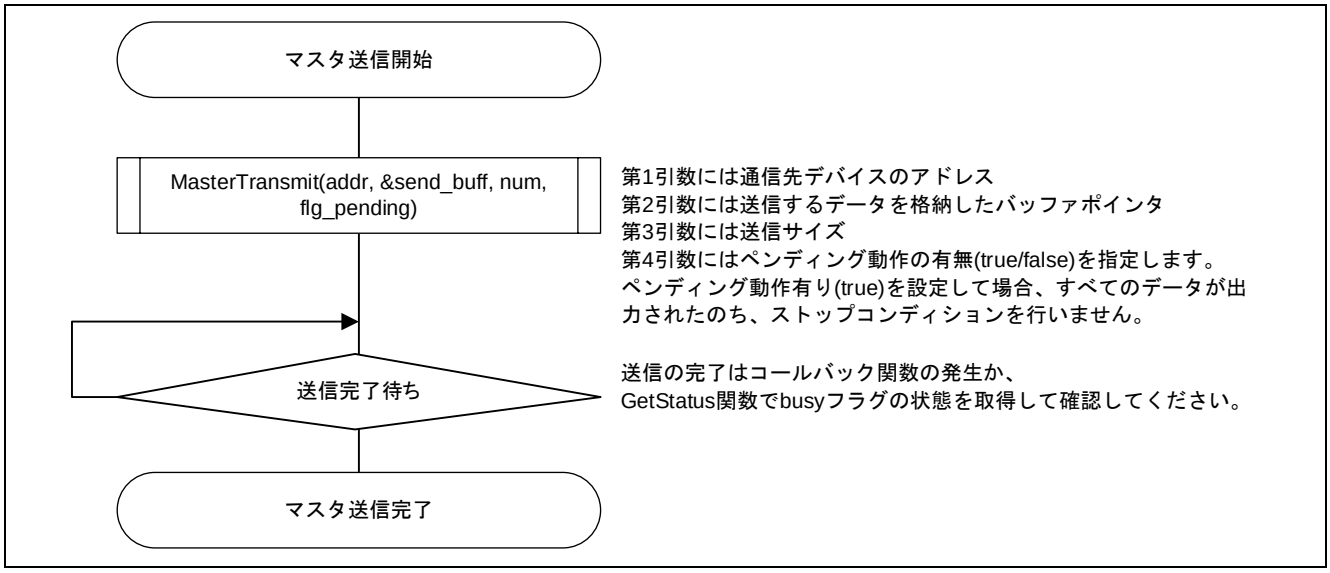

図 3-2 マスタ送信手順

コールバック関数を設定していた場合、マスタ送信が完了するとイベント情報を引数にコールバック関数が呼び出されます。

マスタ送信で発生するイベント情報を表 3-1 に示します。

表 3-1 マスタ送信で発生するイベント情報

イベント情報	内容
ARM_I2C_EVENT_TRANSFER_DONE	正常にマスタ送信が完了しました
ARM_I2C_EVENT_TRANSFER_DONE + ARM_I2C_EVENT_ADDRESS_NACK + ARM_I2C_EVENT_TRANSFER_INCOMPLETE	アドレス送信時に NACK を受信しました（注）
ARM_I2C_EVENT_TRANSFER_DONE + ARM_I2C_EVENT_TRANSFER_INCOMPLETE	データを送信時に NACK を受信して送信を完了しました（注）
ARM_I2C_EVENT_ARBITRATION_LOST + ARM_I2C_EVENT_TRANSFER_DONE + ARM_I2C_EVENT_TRANSFER_INCOMPLETE	アービトレーションロストが発生しました

注　NACK を受信した場合、ペンディング動作を有効にしていた場合でもストップコンディションが出力されます。

マスタ送信処理では TXI 割り込み処理、TEI 割り込み処理、EEI 割り込み処理を使用して通信を実現しています。図 3-3 にマスタ送信動作を示します。

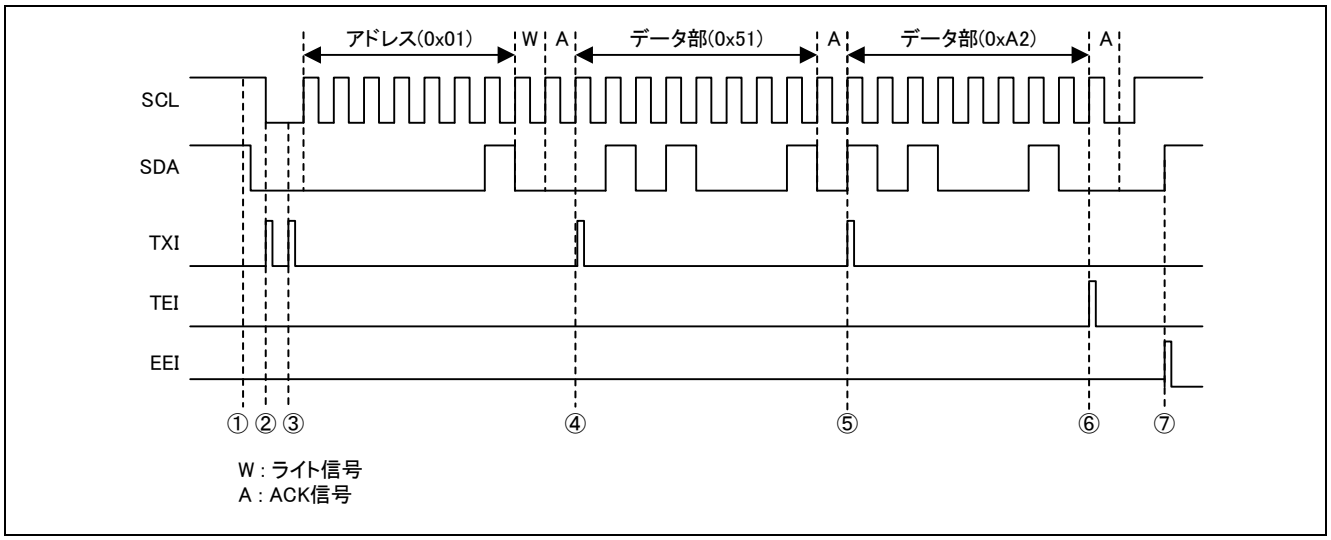

図 3-3 マスタ送信動作（2 バイト送信）

① MasterTransmit 関数を実行すると、スタートコンディションが出力されます

② TXI 割り込みにてアドレスデータを送信データレジスタ（ICDRT）に書き込みます

③ 2 回目の TXI 割り込みにて送信データの 1 バイト目を ICDRT レジスタに書き込みます

④ TXI 割り込みの発生ごとに送信データを順次 ICDRT レジスタに書き込みます

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

⑥ 最終データが出力完了したとき、TEI 割り込みが発生し、ストップコンディション出力を行います

⑦ ストップコンディションが出力されると EEI 割り込みが発生し、コールバック関数を呼び出します

注1. NACK を受信した場合、EEI 割り込みが発生します。EEI 割り込みにて送信処理を中断し、コールバック関数を呼び出します。

注2. アービトレーションロストが発生した場合、その時点で送信を中断してコールバック関数を呼び出します。

3.3　マスタ受信

マスタ受信を開始すると、スタートコンディションを出力したのち、転送先デバイスのアドレス(R)信号を出力したのち、受信動作を開始します。指定したサイズのデータを受信するとストップコンディションを出力してマスタ受信を終了します。

I2C でマスタ受信を行う手順を図 3-4 に示します。

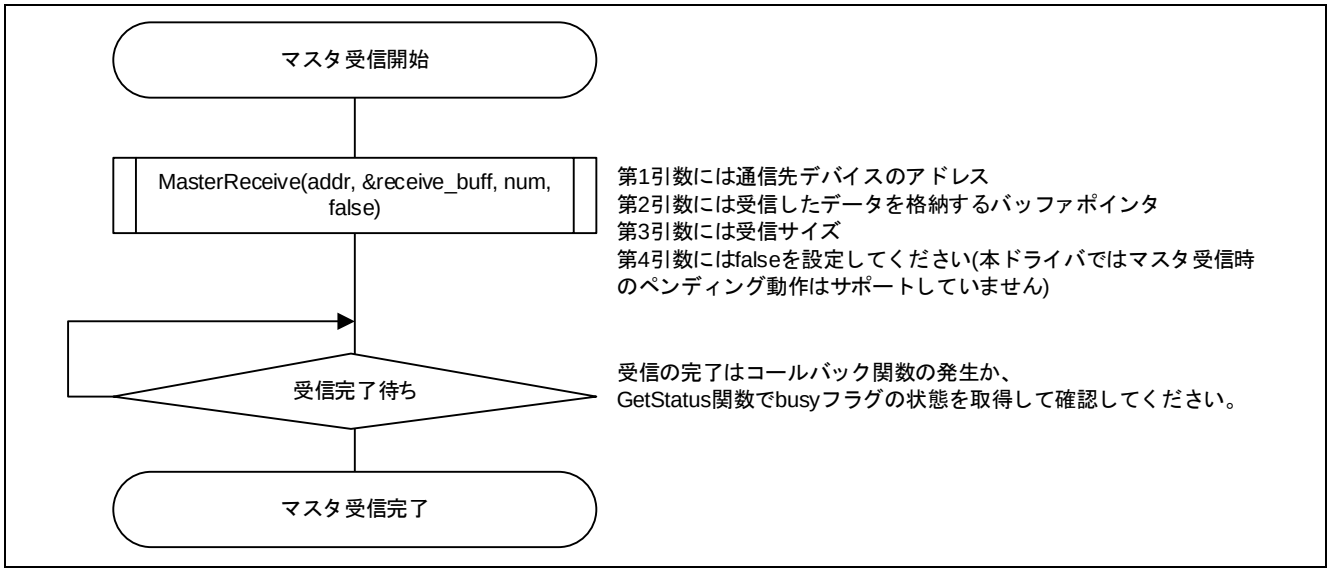

図 3-4 マスタ受信手順

コールバック関数を設定していた場合、マスタ受信が完了するとイベント情報を引数にコールバック関数が呼び出されます。

マスタ受信で発生するイベント情報を表 3-2 に示します。

表 3-2 マスタ受信で発生するイベント情報

イベント情報	内容
ARM_I2C_EVENT_TRANSFER_DONE	正常にマスタ送信が完了しました
ARM_I2C_EVENT_TRANSFER_DONE + ARM_I2C_EVENT_ADDRESS_NACK + ARM_I2C_EVENT_TRANSFER_INCOMPLETE	アドレス送信時に NACK を受信しました
ARM_I2C_EVENT_ARBITRATION_LOST + ARM_I2C_EVENT_TRANSFER_DONE + ARM_I2C_EVENT_TRANSFER_INCOMPLETE	アービトレーションロストが発生しました

マスタ受信処理では TXI 割り込み処理、RXI 割り込み処理、EEI 割り込み処理を使用して通信を実現しています。図 3-5 にマスタ受信動作を示します。

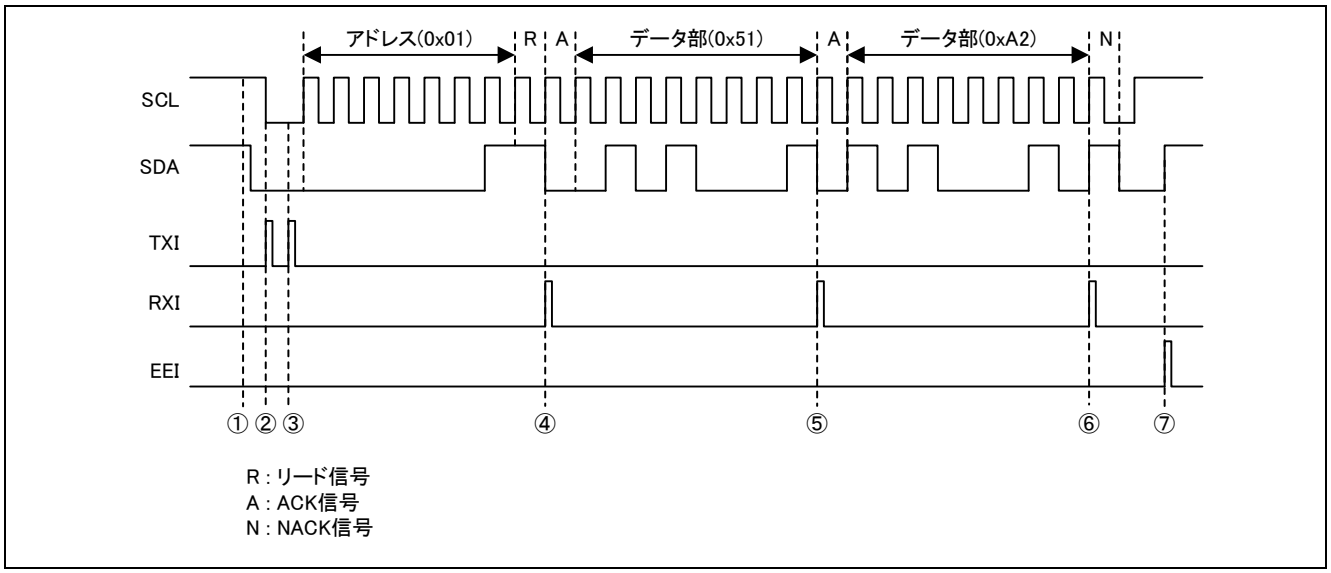

図 3-5 マスタ受信動作（2 バイト受信）

① MasterReceive 関数を実行すると、スタートコンディションが出力されます

② TXI 割り込みにてアドレスデータを送信データレジスタ（ICDRT）に書き込みます

③ 2 回目の TXI 割り込みでは何も処理をせず割り込み処理を終了します

④ リード信号による RXI 割り込みにて、受信データレジスタ（ICDRR）をダミーリードします

⑤ 2 回目の RXI 割り込みで 1 バイト目の受信データを受信バッファに格納します

⑥ 最終データを受信したとき、受信データの格納、NACK 応答を行ったのち、ストップコンディション出力を行います

⑦ ストップコンディションが出力されると EEI 割り込みが発生し、コールバック関数を呼び出します

注1. アドレス部にて NACK を受信した場合、EEI 割り込みが発生します。EEI 割り込みにて受信処理を中断し、コールバック関数を呼び出します。

注2. アービトレーションロストが発生した場合、その時点で送信を中断してコールバック関数を呼び出します。

3.4　スレーブ送信

スレーブ送信を開始するとスレーブ送信待ち状態となります。マスタから自身のアドレスに対するリード信号を受信すると、スレーブ送信を開始します。

I2C でスレーブ送信を行う手順を図 3-6 に示します。

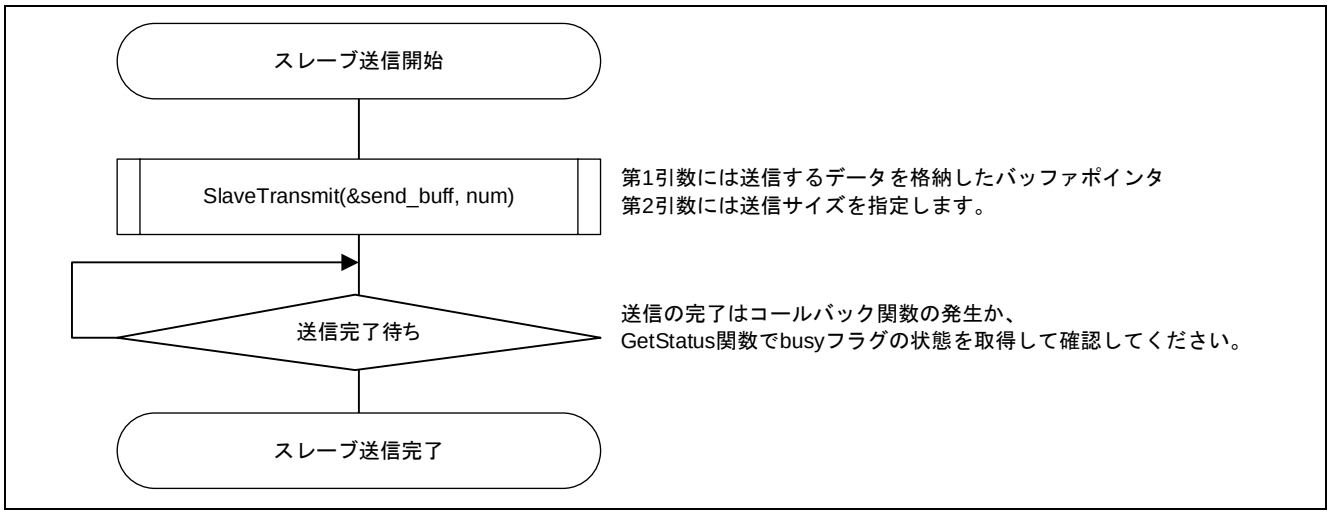

図 3-6 スレーブ送信手順

コールバック関数を設定していた場合、スレーブ送信開始時、およびスレーブ送信完了時にイベント情報を引数にコールバック関数が呼び出されます。

スレーブ送信で発生するイベント情報を表 3-3 に示します。

表 3-3 スレーブ送信で発生するイベント情報

イベント情報	内容
ARM_I2C_EVENT_SLAVE_TRANSMIT	スレーブ送信を開始しました
ARM_I2C_EVENT_TRANSFER_DONE	正常にスレーブ送信が完了しました
ARM_I2C_EVENT_TRANSFER_DONE + ARM_I2C_EVENT_TRANSFER_INCOMPLETE	送信が正常に終了しませんでした(注)
ARM_I2C_EVENT_ARBITRATION_LOST + ARM_I2C_EVENT_TRANSFER_DONE + ARM_I2C_EVENT_TRANSFER_INCOMPLETE	アービトレーションロストが発生しました
ARM_I2C_EVENT_SLAVE_RECEIVE	スレーブ受信要求を受け付けました

注　以下のいずれかの条件が発生した場合に異常と判定します
- 指定したサイズ送信前にストップコンディションを検出した場合
- データ送信中に NACK を検出した場合
  （NACK を検出したのち、クロックが供給された場合は 0xFF を送信します。）
- アドレス受信時に W 信号（受信要求）を受けた場合
  （ジェネラルコールによる受信要求の場合、イベント信号に ARM_I2C_EVENT_GENERAL_CALL も付加されます）

スレーブ送信処理では TXI 割り込み処理、TEI 割り込み処理、EEI 割り込み処理を使用して通信を実現しています。図 3-7 にスレーブ送信動作を示します。

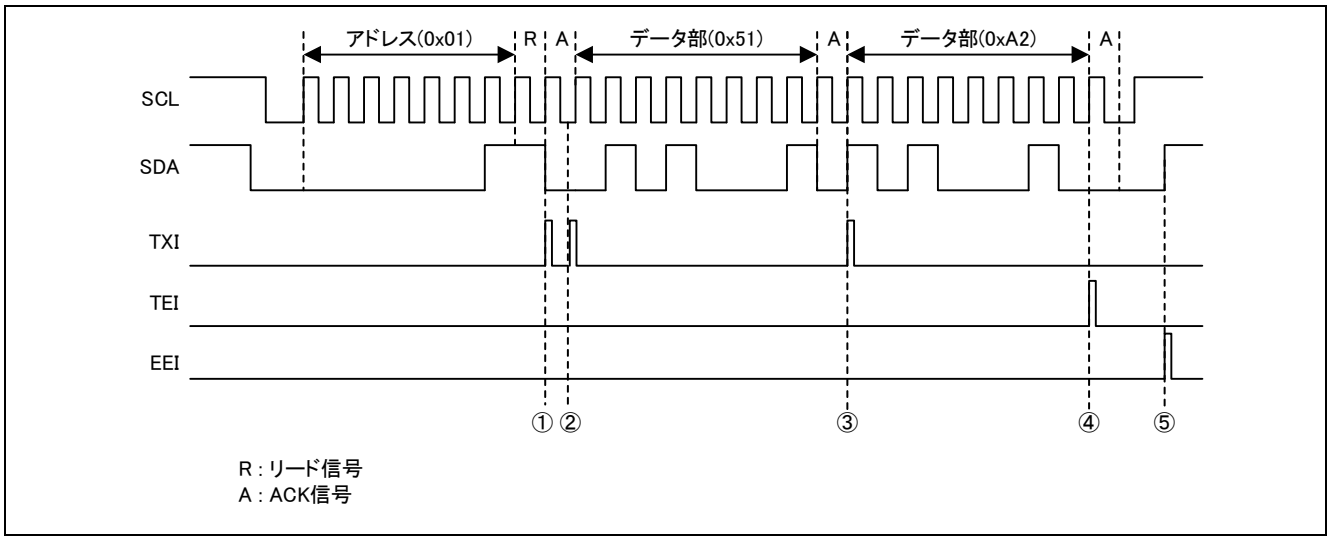

図 3-7 スレーブ送信動作（2 バイト送信）

① SlaveTransmit 関数を実行したのち、自身へのアドレスおよびリード信号を受信すると TXI 割り込みが発生し、送信データの 1 バイト目を送信データレジスタ（ICDRT）に書き込みます
また、コールバック関数を呼び出します

② TXI 割り込みの発生ごとに送信データを順次 ICDRT レジスタに書き込みます

③ 最終データ書き込み後の TXI 割り込みでは、TXI 割り込みを禁止に、TEI 割り込みを許可にします

④ 最終データが出力完了したとき、TEI 割り込みが発生します

⑤ マスタよりストップコンディションが出力されると、コールバック関数が呼び出されます

注1. アドレス部にて W 信号を受信した場合、RXI 割り込みが発生し、コールバック関数を呼び出します。以降の受信データはダミーリードのみ行い、ストップコンディション検出時のコールバック関数にてエラーを返します。

注2. データ送信中に NACK を受信した場合、EEI 割り込みが発生し送信処理を中断します。以降、クロックが入力されても 0xFF のみを送信します。

注3. アービトレーションロストが発生した場合、その時点で送信を中断してコールバック関数を呼び出します。

3.5　スレーブ受信

スレーブ受信を開始するとスレーブ受信待ち状態となります。マスタから自身のアドレスに対するライト信号を受信すると、スレーブ受信を開始します。

I2C でスレーブ受信を行う手順を図 3-8 に示します。

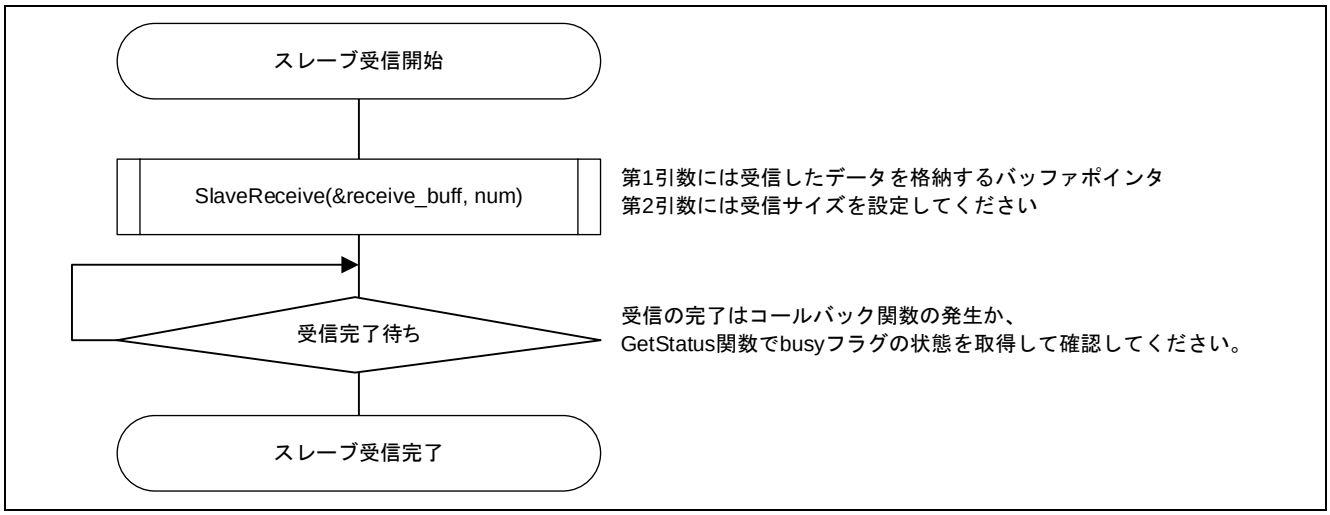

図 3-8 スレーブ受信手順

コールバック関数を設定していた場合、スレーブ受信開始時、およびスレーブ受信完了時にイベント情報を引数にコールバック関数が呼び出されます。

スレーブ受信で発生するイベント情報を表 3-4 に示します。

表 3-4 スレーブ受信で発生するイベント情報

イベント情報	内容
ARM_I2C_EVENT_SLAVE_RECEIVE	スレーブ受信を開始しました
ARM_I2C_EVENT_TRANSFER_DONE	正常にスレーブ受信が完了しました
ARM_I2C_EVENT_TRANSFER_DONE+ ARM_I2C_EVENT_GENERAL_CALL	正常にジェネラルコールによる受信が完了しました
ARM_I2C_EVENT_TRANSFER_DONE + ARM_I2C_EVENT_TRANSFER_INCOMPLETE	受信が正常に終了しませんでした(注)
ARM_I2C_EVENT_SLAVE_TRANSMIT	スレーブ送信要求を受け付けました

注　以下のいずれかの条件が発生した場合に異常と判定します
- 指定したサイズ受信前にストップコンディションを検出した場合
  （ジェネラルコールによる受信の場合、イベント信号に ARM_I2C_EVENT_GENERAL_CALL も付加されます）
- アドレス受信時に R 信号（送信要求）を受けた場合

スレーブ受信処理では RXI 割り込み処理、EEI 割り込み処理を使用して通信を実現しています。図 3-9 にスレーブ受信動作を示します。

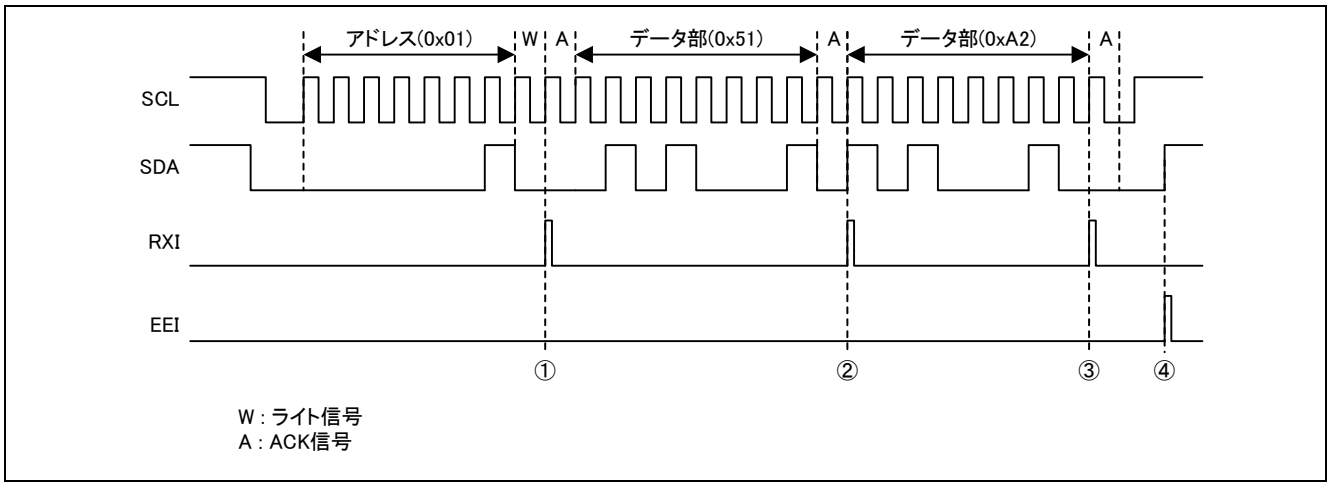

図 3-9 スレーブ受信動作（2 バイト受信）

① SlaveReceive 関数を実行したのち、自身へのアドレスおよびライト信号を受信すると RXI 割り込みが発生します。（最初の受信データは読み捨てます）
また、コールバック関数を呼び出します

② RXI 割り込みの発生ごとに受信データを順次指定された受信バッファに取り込みます

③ 最終データ受信時の RXI 割り込みで、NACK 出力設定を行います
（以降、指定サイズ以上のデータを受信した場合は、データを読み捨て、NACK を応答します）

④ マスタよりストップコンディションが出力されると、コールバック関数が呼び出されます

注　アドレス部にて R 信号を受信した場合、TXI 割り込みが発生し、コールバック関数を呼び出します。以降、クロックが入力されて場合は 0xFF のみを送信し、ストップコンディション検出時のコールバック関数にてエラーを返します。

3.6　コンフィグレーション

I2C 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_i2c_cfg.h ファイルに用意します。

3.6.1　ノイズフィルタ定義

ノイズフィルタ回路を有効もしくは無効に設定します。有効に設定する場合はフィルタ段数を指定します。

名称：RIIC_NOISE_FILTER

表 3-5 RIIC_NOISE_FILTER の設定

設定値	内容
0	ノイズフィルタ無効
1	1IICφ サイクル以下のノイズを除去（フィルタは 1 段）
2（初期値）	2IICφ サイクル以下のノイズを除去（フィルタは 2 段）
3	3IICφ サイクル以下のノイズを除去（フィルタは 3 段）
4	4IICφ サイクル以下のノイズを除去（フィルタは 4 段）

3.6.2　バス速度自動計算 有効/無効定義

バス速度の自動計算を有効もしくは無効に設定します。

名称：RIIC_BUS_SPEED_CAL_ENABLE

表 3-6 RIIC_BUS_SPEED_CAL_ENABLE の設定

設定値	内容
0	バス速度自動計算を無効にします
1（初期値）	バス速度自動計算を有効にします

3.6.3　SCL 立ち上がり時間/立ち下がり時間定義

バス速度の自動計算時に使用する SCL 信号の立ち上がり時間、立ち下がり時間を設定します。

時間の設定は秒で記載してください。

表 3-7 SCL 立ち上がり時間/立ち下がり時間定義の名称と設定

名称	初期値	内容
RIIC_STAD_SCL_UP_TIME	(1000E-9)	スタンダードモード時の SCL 信号立ち上がり時間（初期値 1000ns）
RIIC_STAD_SCL_DOWN_TIME	(300E-9)	スタンダードモード時の SCL 信号立ち下がり時間（初期値 300ns）
RIIC_FAST_SCL_UP_TIME	(300E-9)	ファストモード時の SCL 信号立ち上がり時間（初期値 300ns）
RIIC_FAST_SCL_DOWN_TIME	(300E-9)	ファストモード時の SCL 信号立ち下がり時間（初期値 300ns）

3.6.4　　バス速度設定定義

バス速度の自動計算無効時に使用するレジスタの設定値を定義します。

表 3-8 バス速度設定定義の名称と設定

名称	初期値	内容
RIIC_STAD_ICBRL	(15)	スタンダードモード時の ICBRL 設定値（0～31）
RIIC_STAD_ICBRH	(12)	スタンダードモード時の ICBRH 設定値（0～31）
RIIC_STAD_CKS	(3)	スタンダードモード時の CKS 設定値（0～7）
RIIC_FAST_ICBRL	(17)	ファストモード時の ICBRL 設定値（0～31）
RIIC_ FAST_ICBRH	(6)	ファストモード時の ICBRH 設定値（0～31）
RIIC_ FAST_CKS	(1)	ファストモード時の CKS 設定値（0～7）

3.6.5　　TXI 割り込み優先レベル

TXIn 割り込みの優先レベルを設定します。(n=0、1)

名称：RIIC0_TXI_PRIORITY、RIIC1_TXI_PRIORITY

表 3-9 RIICn_TXI_PRIORITY の設定

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

3.6.6　　TEI 割り込み優先レベル

TEIn 割り込みの優先レベルを設定します。(n=0、1)

名称：RIIC0_TEI_PRIORITY、RIIC1_TEI_PRIORITY

表 3-10 RIICn_TEI_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

3.6.7　RXI 割り込み優先レベル

RXIn 割り込みの優先レベルを設定します。(n=0、1)

名称：RIIC0_RXI_PRIORITY、RIIC1_RXI_PRIORITY

表 3-11 RIICn_RXI_PRIORITY の設定

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

3.6.8　EEI 割り込み優先レベル

EEIn 割り込みの優先レベルを設定します。(n=0、1)

名称：RIIC0_EEI_PRIORITY、RIIC1_EEI_PRIORITY

表 3-12 RIICn_EEI_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

3.6.9　関数の RAM 配置

I2C 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：RIIC_CFG_SECTION_xxx

xxx には関数名をすべて大文字で記載

例）　ARM_I2C_INITIALIZE 関数　→　RIIC_CFG_SECTION_ARM_I2C_INITIALIZE

表 3-13 RIIC_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 3-14 各関数の RAM 配置初期状態

番号	関数名	RAM 配置
1	ARM_I2C_Initialize	
2	ARM_I2C_Uninitialize	
3	ARM_I2C_PowerControl	
4	ARM_I2C_MasterTransmit	
5	ARM_I2C_MasterReceive	
6	ARM_I2C_SlaveTransmit	
7	ARM_I2C_SlaveReceive	
8	ARM_I2C_GetDataCount	
9	ARM_I2C_Control	
10	ARM_I2C_GetStatus	
11	ARM_I2C_GetVersion	
12	ARM_I2C_GetCapabilities	
13	iic_txi_interrupt（TXI 割り込み処理）	✔
14	iic_tei_interrupt（TEI 割り込み処理）	✔
15	iic_rxi_interrupt（RXI 割り込み処理）	✔
16	iic_eei_interrupt（EEI 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

I2C 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。

また、本章では以下の略称を使用します。

ST：スタートコンディション

SP：ストップコンディション

RS：リスタートコンディション

4.1.1　ARM_I2C_Initialize 関数

表 4-1 ARM_I2C_Initialize 関数仕様

書式	static int32_t ARM_I2C_Initialize(ARM_I2C_SignalEvent_t cb_event, st_i2c_resources_t *p_i2c)
仕様説明	I2C ドライバの初期化（RAM の初期化、レジスタ設定、端子設定、NVIC への登録）を行います 初期状態では通信速度 100kbps で RIIC を初期化します
引数	ARM_I2C_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_i2c_resources_t *p_i2c : I2C のリソース 初期化する I2C のリソースを指定します。
戻り値	ARM_DRIVER_OK　　I2C の初期化成功
	ARM_DRIVER_ERROR　　I2C 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 ・r_system_cfg.h で TXI、RXI、TEI、EEI 割り込みが未使用定義 （SYSTEM_IRQ_EVENT_NUMBER_NOT_USED）になっている場合 ・r_i2c_cfg.h で RIICn_TXI_PRIORITY の設定が定義範囲を超えている場合 ・r_i2c_cfg.h で RIICn_RXI_PRIORITY の設定が定義範囲を超えている場合 ・r_i2c_cfg.h で RIICn_TEI_PRIORITY の設定が定義範囲を超えている場合 ・r_i2c_cfg.h で RIICn_EEI_PRIORITY の設定が定義範囲を超えている場合 ・使用する RIIC 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 RIICn 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 I2C 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);
	
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	
	main()
	{
	　　I2cDev0->Initialize(callback);
	}

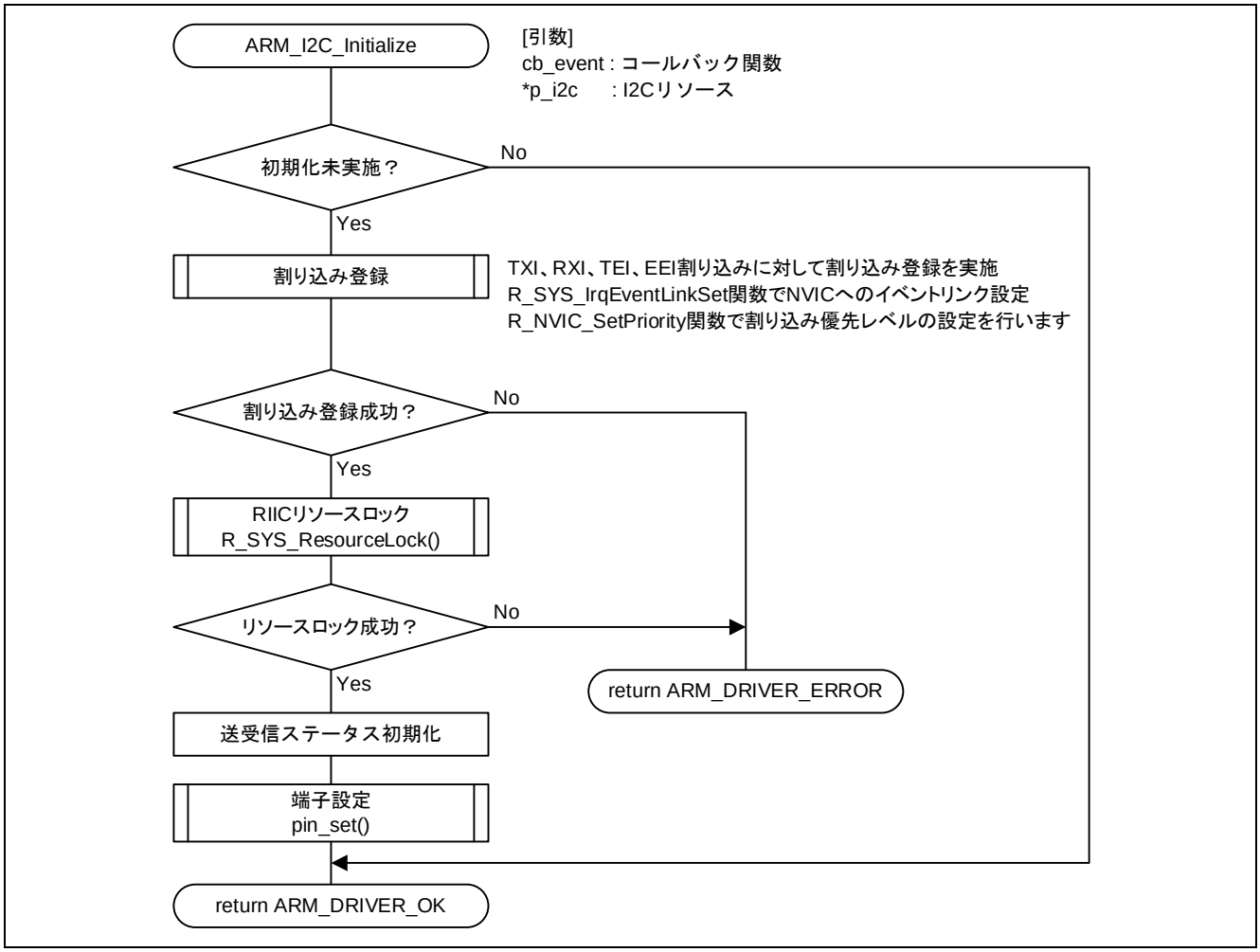

図 4-1 ARM_I2C_Initialize 関数処理フロー

4.1.2　ARM_I2C_Uninitialize 関数

表 4-2 ARM_I2C_Uninitialize 関数仕様

書式	static int32_t ARM_I2C_Uninitialize(st_i2c_resources_t *p_i2c)
仕様説明	I2C ドライバを解放します
引数	st_i2c_resources_t *p_i2c : I2C のリソース 解放する I2C のリソースを指定します。
戻り値	ARM_DRIVER_OK　　I2C の解放成功

備考	インスタンスからのアクセス時は I2C 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]
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	
	main()
	{
	I2cDev0->Uninitialize();
	}

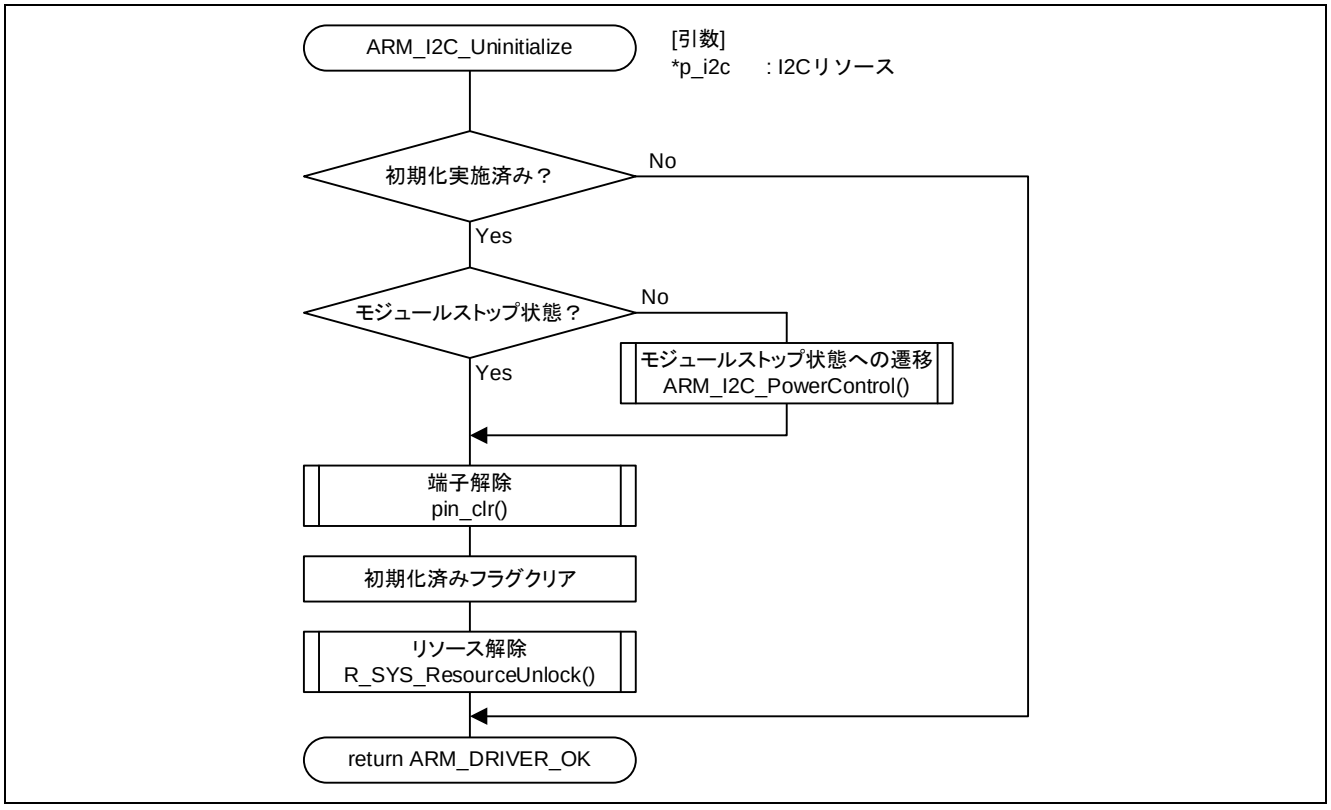

図 4-2 ARM_USART_Uninitailze 関数処理フロー

4.1.3　ARM_I2C_PowerControl 関数

表 4-3 ARM_I2C_PowerControl 関数仕様

書式	static int32_t ARM_I2C_PowerControl(ARM_POWER_STATE state, st_i2c_resources_t *p_i2c)
仕様説明	I2C 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_i2c_resources_t *p_i2c : I2C のリソース 電源供給する I2C 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 ・I2C 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） ・100kbps の通信設定に失敗した場合（PCLKB の上限 32MHz 以下で動作する場合、設定失敗することはありません）
	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 I2C 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]
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	
	main()
	{
	　　I2cDev0->Uninitialize();
	}

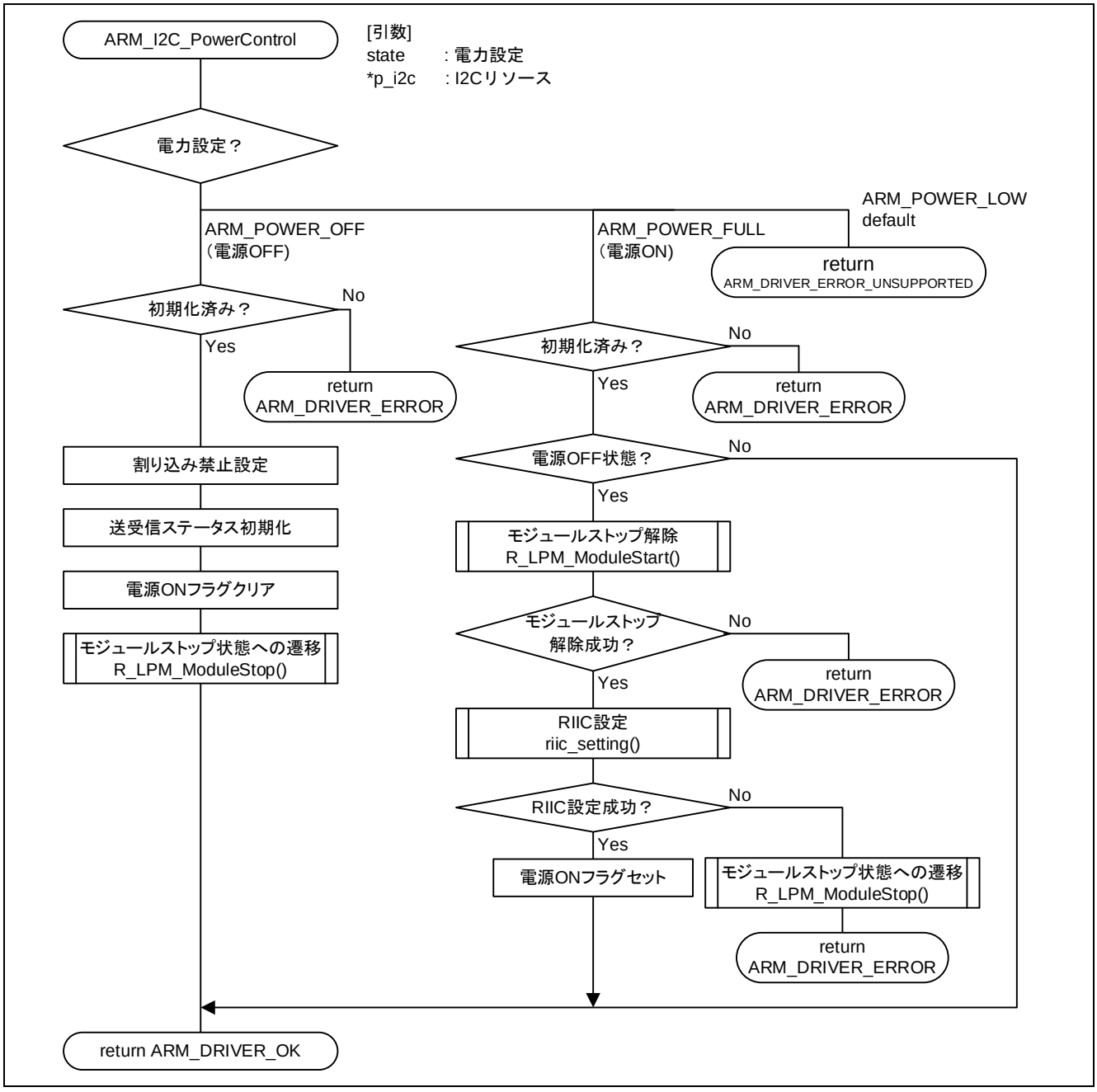

図 4-3 ARM_I2C_PowerControl 関数処理フロー

4.1.4　ARM_I2C_MasterTransmit 関数

表 4-4 ARM_I2C_MasterTransmit 関数仕様

書式	static int32_t ARM_I2C_MasterTransmit(uint32_t addr, const uint8_t *data, uint32_t num, bool xfer_pending, st_i2c_resources_t *p_i2c)
仕様説明	マスタ送信を開始します
引数	uint32_t addr : 送信先デバイスのアドレス RIIC_ADDR_NONE を指定した場合、ST と SP のみ出力します
	const uint8_t *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 送信先デバイスのアドレスに RIIC_ADDR_NONE、または送信サイズを 0 にする場合は、NULL を指定してください
	uint32_t num : 送信サイズ 送信するデータサイズを指定します 0 を設定した場合は、ST、アドレス、SP のみ送信されます
	bool xfer_pending : ペンディングモード動作の設定 true : ペンディングモード有効（送信完了後、ストップコンディションを出力しない） false : ペンディングモード無効（送信完了後、ストップコンディションを出力する）
	st_i2c_resources_t *p_i2c : I2C のリソース 送信する I2C のリソースを指定します。
戻り値	ARM_DRIVER_OK　マスタ送信開始成功
	ARM_DRIVER_ERROR　マスタ送信失敗 電源 OFF 状態の場合で実行した場合、マスタ送信失敗となります
	ARM_DRIVER_ERROR_BUSY　ビジー状態による送信失敗 以下のいずれかの状態を検出するとビジー状態による送信失敗となります ・ステータスによる送信中判定（status.busy == 1） ・スレーブ状態でのバスビジー判定（ICCR2.MST=0, ICCR2.BBSY=1）
	ARM_DRIVER_ERROR_PARAMETER　パラメータエラー 送信サイズが 1 以上で、送信バッファが NULL の場合、パラメータエラーとなります

備考	インスタンスからのアクセス時は I2C 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]
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	const uint8_t tx_data[2] = {0x51, 0xA2};
	
	main()
	{
	　　I2cDev0->MasterTransmit(0x01,　&tx_data[0], 2, false);
	}

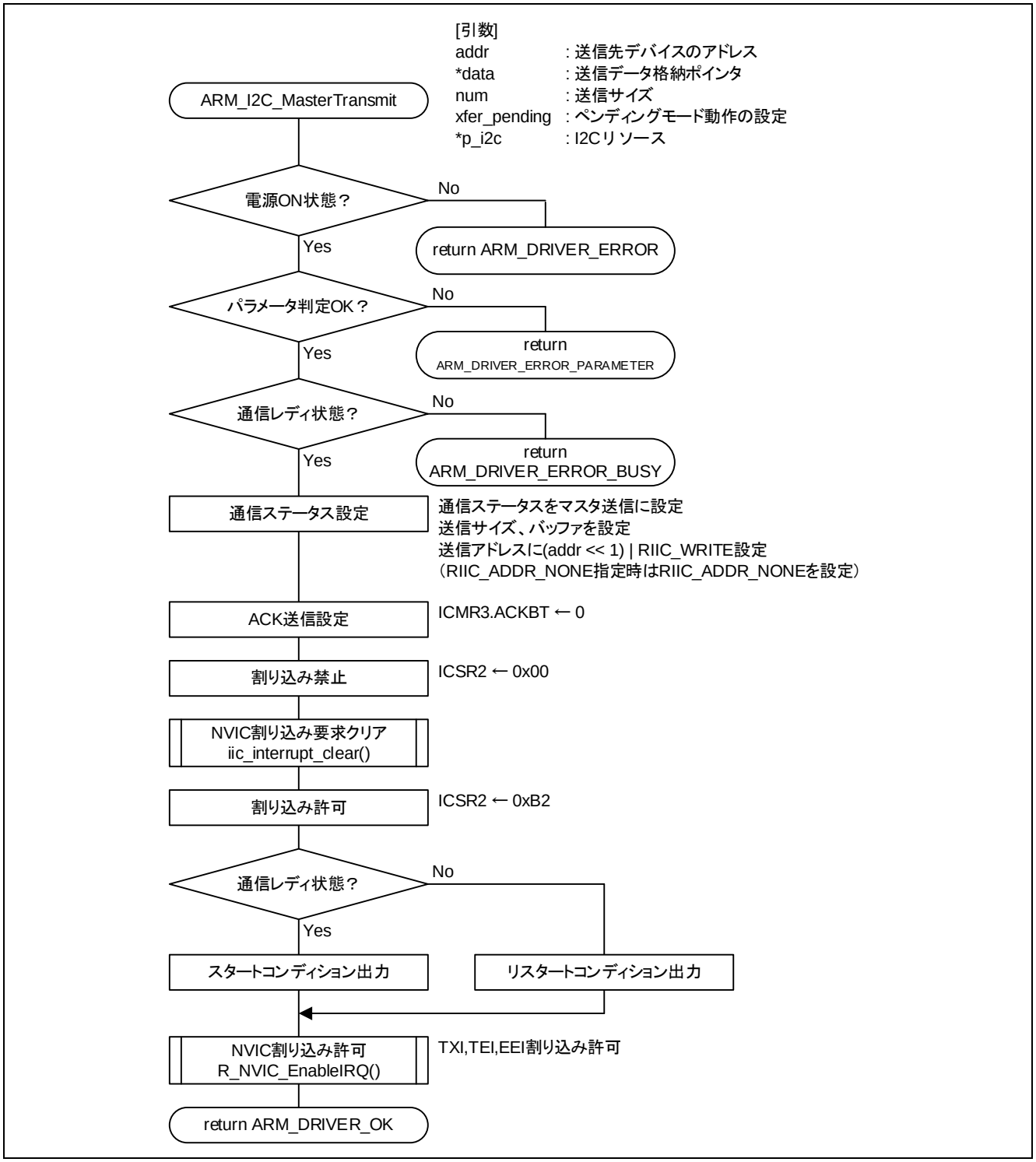

図 4-4 ARM_I2C_MasterTransmit 関数処理フロー

4.1.5　ARM_I2C_MasterReceive 関数

表 4-5 ARM_I2C_MasterReceive 関数仕様

書式	static int32_t ARM_I2C_MasterReceive(uint32_t addr, uint8_t *data, uint32_t num, bool xfer_pending, st_i2c_resources_t *p_i2c)
仕様説明	マスタ受信を開始します
引数	uint32_t addr : 通信先デバイスのアドレス
	const uint8_t *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
	bool xfer_pending : ペンディングモード動作の設定（false のみ有効） false : ペンディングモード無効（受信完了後、ストップコンディションを出力する）
	st_i2c_resources_t *p_i2c : I2C のリソース 受信する I2C のリソースを指定します。
戻り値	ARM_DRIVER_OK　マスタ受信開始成功
	ARM_DRIVER_ERROR　マスタ受信失敗 電源 OFF 状態の場合で実行した場合、マスタ受信失敗となります
	ARM_DRIVER_ERROR_BUSY　ビジー状態による受信失敗 以下のいずれかの状態を検出するとビジー状態による受信失敗となります ・ステータスによる送信中判定（status.busy == 1） ・スレーブ状態でのバスビジー判定（ICCR2.MST=0, ICCR2.BBSY=1）
	ARM_DRIVER_ERROR_PARAMETER　パラメータエラー 以下のいずれかの条件を検出するとパラメータエラーとなります ・xfer_pending に true を設定した場合 ・通信先デバイスに RIIC_ADDR_NONE を設定した場合 ・受信データ格納ポインタが NULL の場合 ・受信サイズが 0 の場合

備考	インスタンスからのアクセス時は I2C 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]
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	uint8_t rx_data[2];
	
	main()
	{
	　　I2cDev0->MasterReceive(0x01,　&rx_data[0], 2, false);
	}

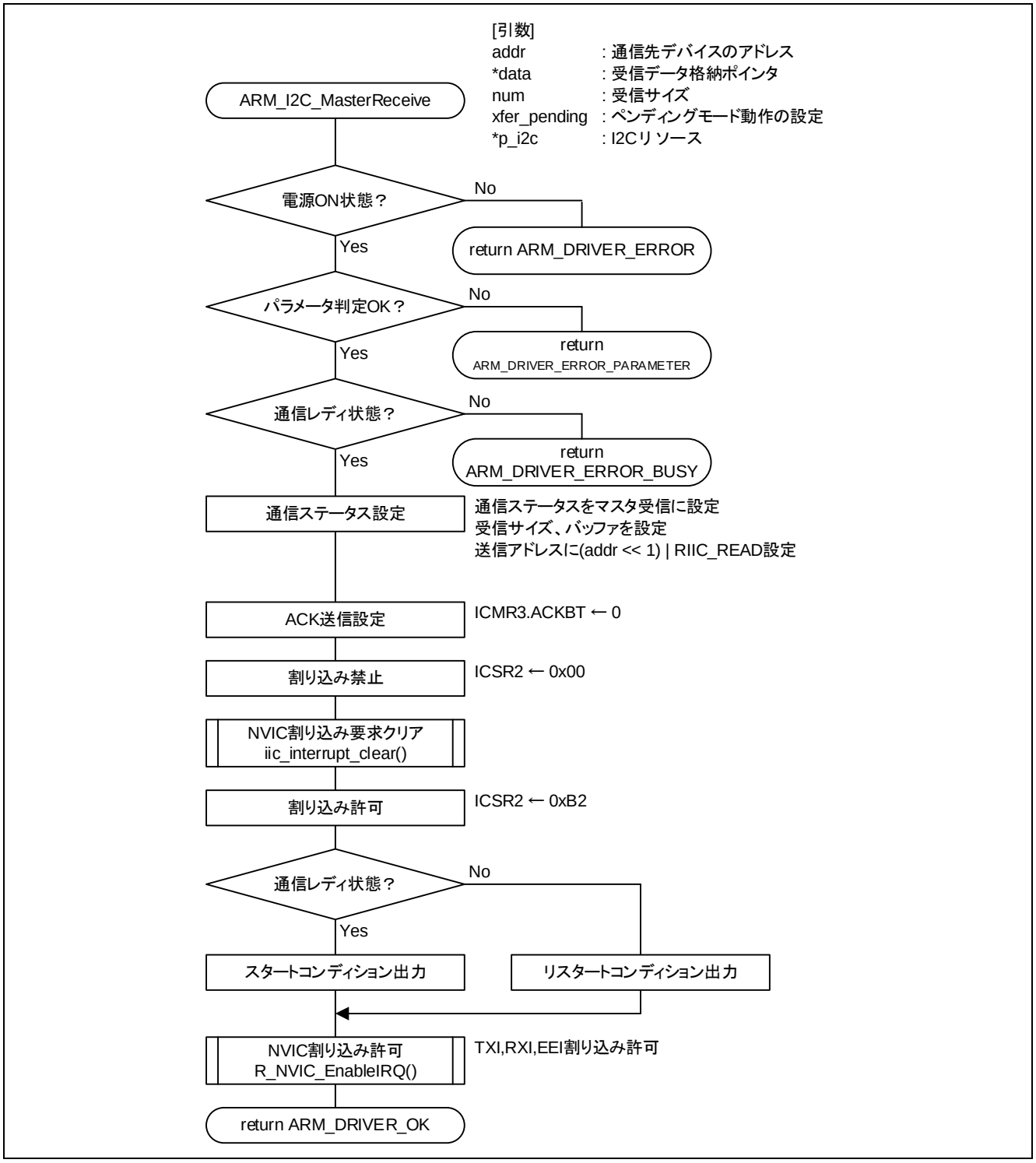

図 4-5 ARM_I2C_MasterReceive 関数処理フロー

4.1.6　ARM_I2C_SlaveTransmit 関数

表 4-6 ARM_I2C_SlaveTransmit 関数仕様

書式	static int32_t ARM_I2C_SlaveTransmit(const uint8_t *data, uint32_t num, st_i2c_resources_t *p_i2c)
仕様説明	スレーブ送信を開始します
引数	const uint8_t *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_i2c_resources_t *p_i2c : I2C のリソース 送信する I2C のリソースを指定します。
戻り値	ARM_DRIVER_OK　スレーブ送信開始成功
	ARM_DRIVER_ERROR　スレーブ送信失敗 電源 OFF 状態の場合で実行した場合、マスタ送信失敗となります
	ARM_DRIVER_ERROR_BUSY　ビジー状態による送信失敗 ステータスによる送信中判定（status.busy == 1）の場合に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 以下のいずれかの状態を検出するとパラメータエラーとなります ・送信サイズが 0 の場合 ・送信データ格納ポインタが NULL の場合

備考	インスタンスからのアクセス時は I2C 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]
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	const uint8_t tx_data[2] = {0x51, 0xA2};
	
	main()
	{
	　　I2cDev0->SlaveTransmit(&tx_data[0], 2);
	}

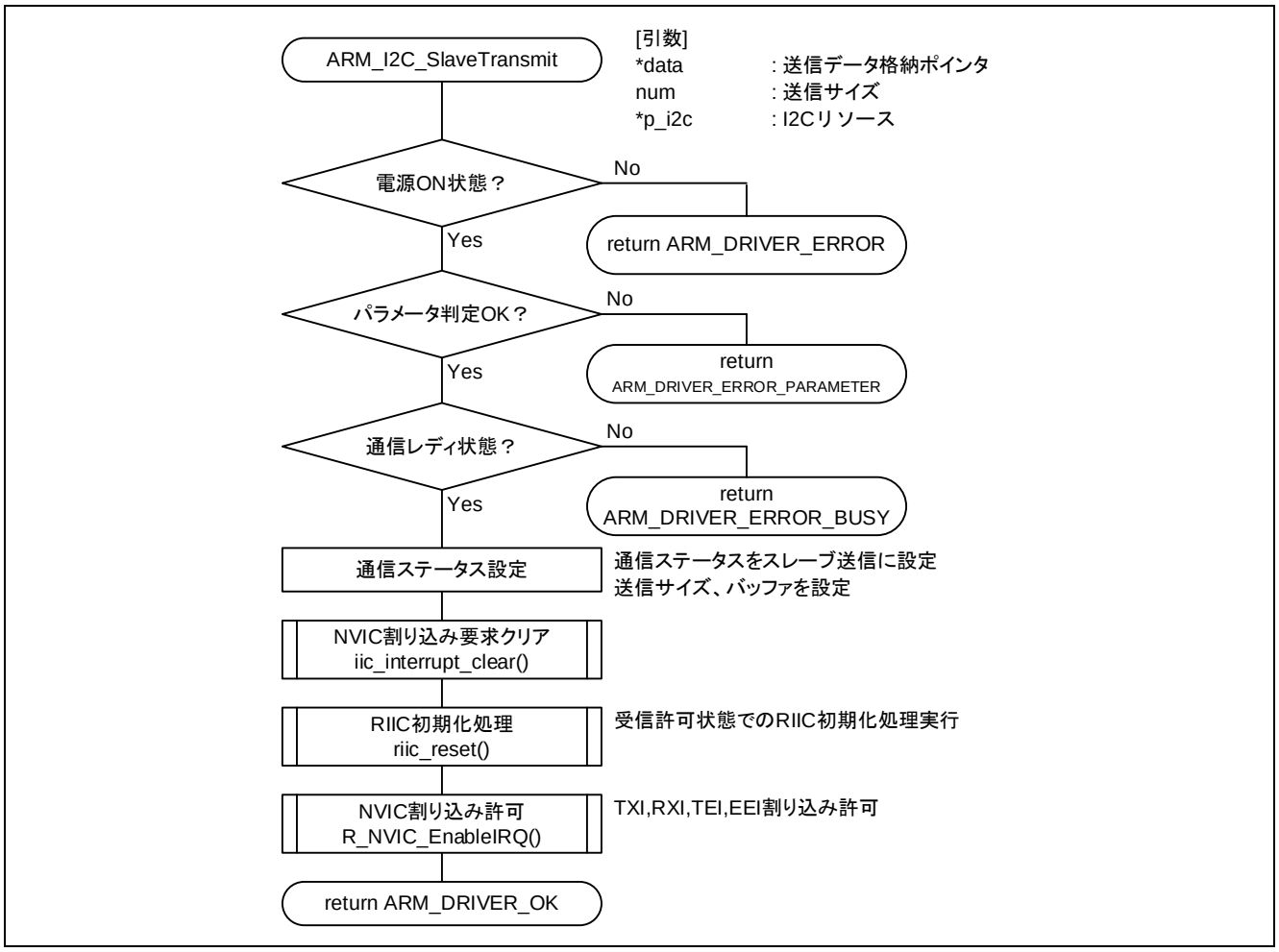

図 4-6 ARM_I2C_SlaveTransmit 関数処理フロー

4.1.7　ARM_I2C_SlaveReceive 関数

表 4-7 ARM_I2C_SlaveReceive 関数仕様

書式	static int32_t ARM_I2C_SlaveReceive(uint8_t *data, uint32_t num, st_i2c_resources_t *p_i2c)
仕様説明	スレーブ受信を開始します
引数	const uint8_t *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_i2c_resources_t *p_i2c : I2C のリソース 受信する I2C のリソースを指定します。
戻り値	ARM_DRIVER_OK　スレーブ受信開始成功
	ARM_DRIVER_ERROR　スレーブ受信失敗 電源 OFF 状態の場合で実行した場合、スレーブ受信失敗となります
	ARM_DRIVER_ERROR_BUSY　ビジー状態による受信失敗 ステータスによる送信中判定（status.busy == 1）の場合、ビジー状態の受信失敗となります
	ARM_DRIVER_ERROR_PARAMETER　パラメータエラー 以下のいずれかの条件を検出するとパラメータエラーとなります ・受信データ格納ポインタが NULL の場合 ・受信サイズが 0 の場合

備考	インスタンスからのアクセス時は I2C 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]
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	uint8_t rx_data[2];
	
	main()
	{
	　　I2cDev0->SlaveReceive(&rx_data[0], 2);
	}

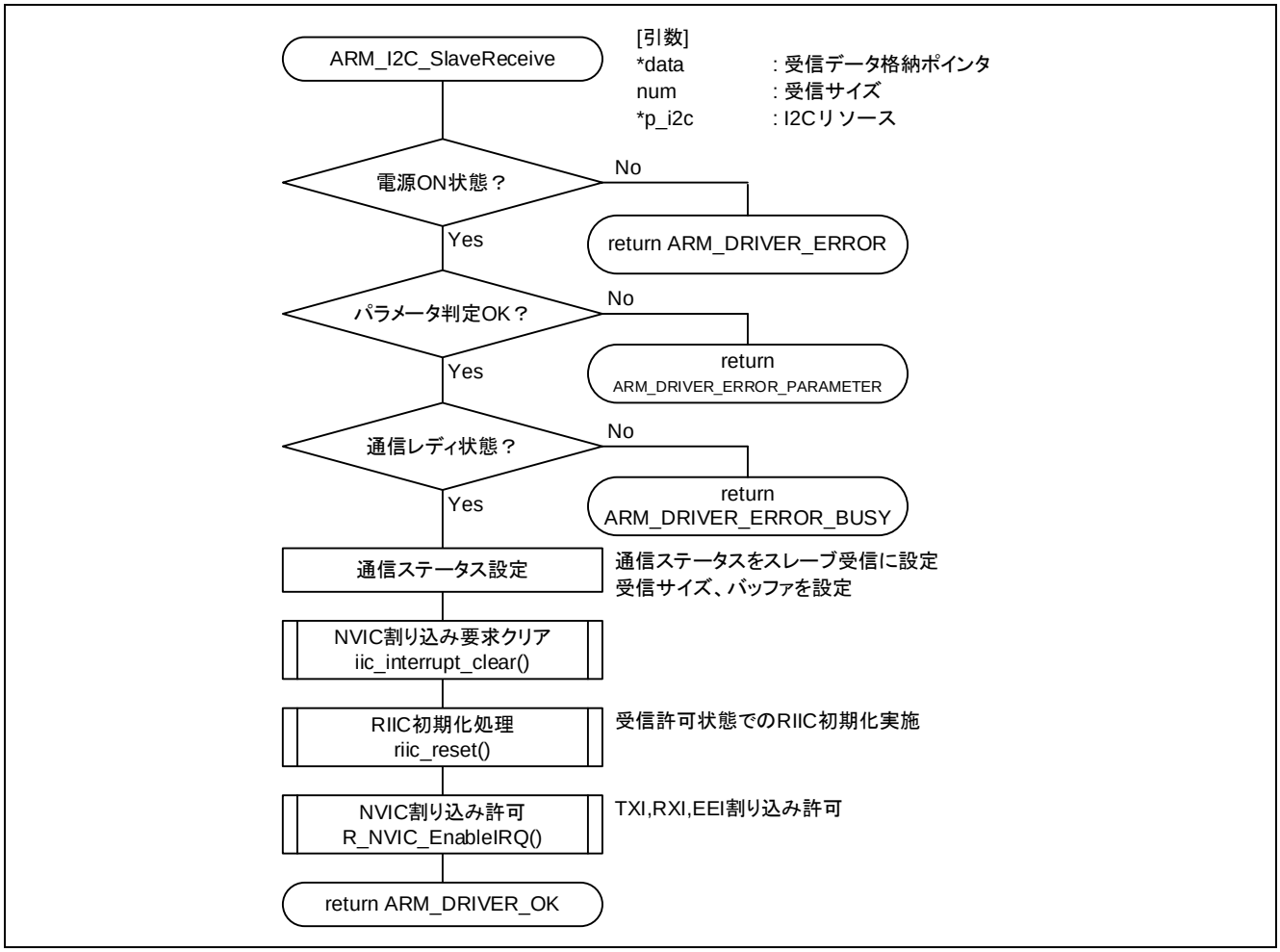

図 4-7 ARM_I2C_SlaveReceive 関数処理フロー

4.1.8　ARM_I2C_GetDataCount 関数

表 4-8 ARM_I2C_GetDataCount 関数仕様

書式	static int32_t ARM_I2C_GetDataCount(st_i2c_resources_t *p_i2c)
仕様説明	送信データ数を取得します
引数	st_i2c_resources_t *p_i2c : I2C のリソース 対象の I2C のリソースを指定します。
戻り値	送信データ数

備考	インスタンスからのアクセス時は I2C 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]
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	
	main()
	{
	　uint32_t snd_cnt;
	　snd_cnt = I2cDev0->GetDataCount();
	}

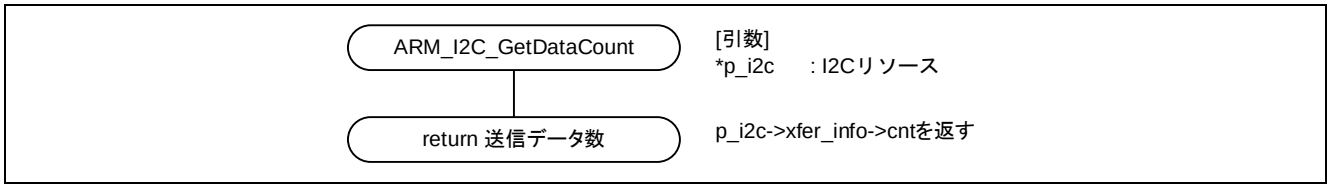

図 4-8 ARM_I2C_GetDataCount 関数処理フロー

4.1.9　ARM_I2C_Control 関数

表 4-9 ARM_I2C_Control 関数仕様

書式	static int32_t ARM_I2C_Control(uint32_t control, uint32_t arg, st_i2c_resources_t *p_i2c)
仕様説明	I2C の制御コマンドを実行します
引数	uint32_t control : 制御コマンド 以下のいずれかの制御コマンドを指定します ・ARM_I2C_OWN_ADDRESS　: 自身のスレーブアドレス設定コマンド ・ARM_I2C_BUS_SPEED　: I2C バス速度設定コマンド ・ARM_I2C_BUS_CLEAR　: バスクリアコマンド ・ARM_I2C_ABORT_TRANSFER　: 送受信中断コマンド
	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 4-10 参照）
	st_i2c_resources_t *p_i2c : I2C のリソース 制御対象の I2C 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 状態の場合で実行した場合、スレーブ受信失敗となります ・バス速度設定コマンドで、指定したバス速度が設定できなかった場合
	ARM_DRIVER_ERROR_BUSY　ビジー状態による制御コマンド実行失敗 通信中に I2C バス速度設定コマンド(ARM_I2C_BUS_SPEED)を実行した場合、ビジー状態による制御コマンド実行失敗となります
	ARM_DRIVER_ERROR_UNSUPPORTED　サポート外の制御コマンド実行失敗 以下のいずれかの条件を検出するとサポート外の制御による制御コマンド実行失敗となります ・制御コマンドに規定外の値を設定した場合 ・I2C バス速度設定コマンド(ARM_I2C_BUS_SPEED)の引数に規定外の値を設定した場合

備考	インスタンスからのアクセス時は I2C 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]
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	
	main()
	{
	　　I2cDev0->Control(ARM_I2C_OWN_ADDRESS, 0x01 | ARM_I2C_ADDRESS_GC);
	　　I2cDev0->Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_FAST);
	}

表 4-10 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_I2C_OWN_ADDRESS	0x00～0x7F (| ARM_I2C_ADDRESS_GC)	自身のスレーブアドレスを設定します ARM_I2C_ADDRESS_GC を含めて指定した場合、ジェネラルコールへの応答も行います
ARM_I2C_BUS_SPEED	ARM_I2C_BUS_SPEED_STANDARD	バス速度をスタンダードスピード(100kbps)に設定します
	ARM_I2C_BUS_SPEED_FAST	バス速度をファストスピード(400kbps)に設定します
ARM_I2C_BUS_CLEAR	NULL	現在の状態により、以下の処理を実行します [マスタモードの場合] SCL を 9 クロック出力します [スレーブモードかつバス解放中] ① ST 出力 ② SCL を 9 クロック出力 ③ SP 出力 [スレーブモードかつ送信中] 送信データを 0xFF にします [スレーブモードかつ受信中] NACK 出力設定にします
ARM_I2C_ABORT_TRANSFER	NULL	送信、または受信動作を中断します

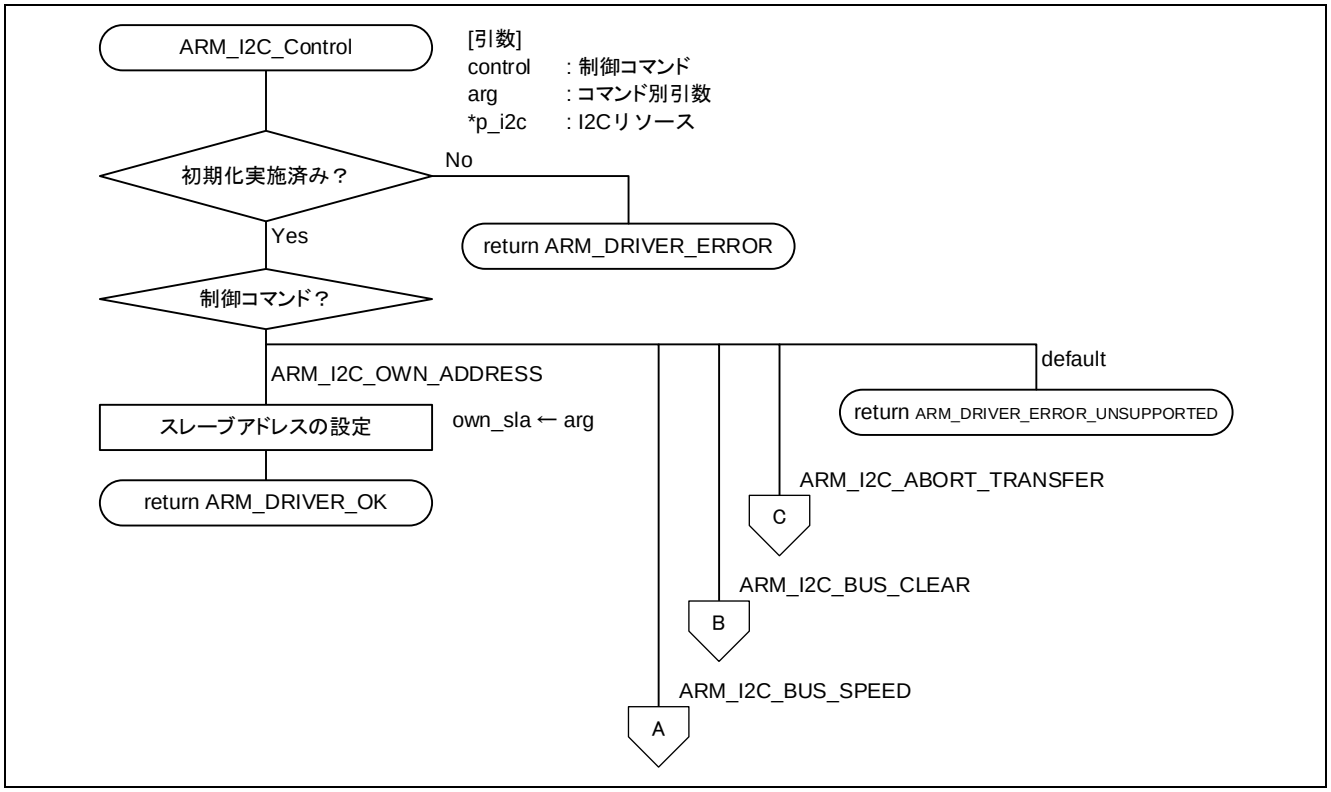

図 4-9 ARM_I2C_Control 関数処理フロー(1/3)

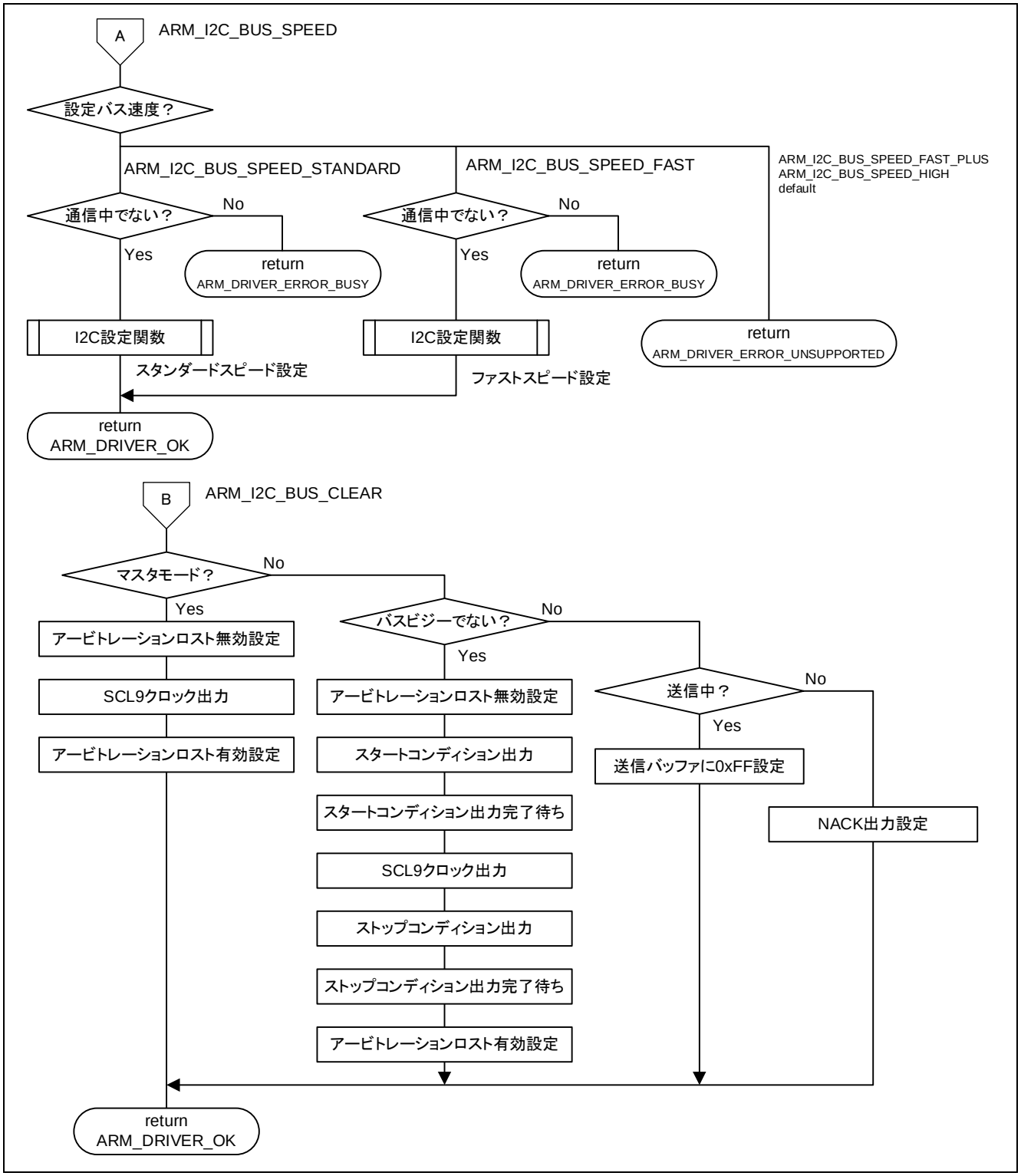

図 4-10 ARM_I2C_Control 関数処理フロー(2/3)

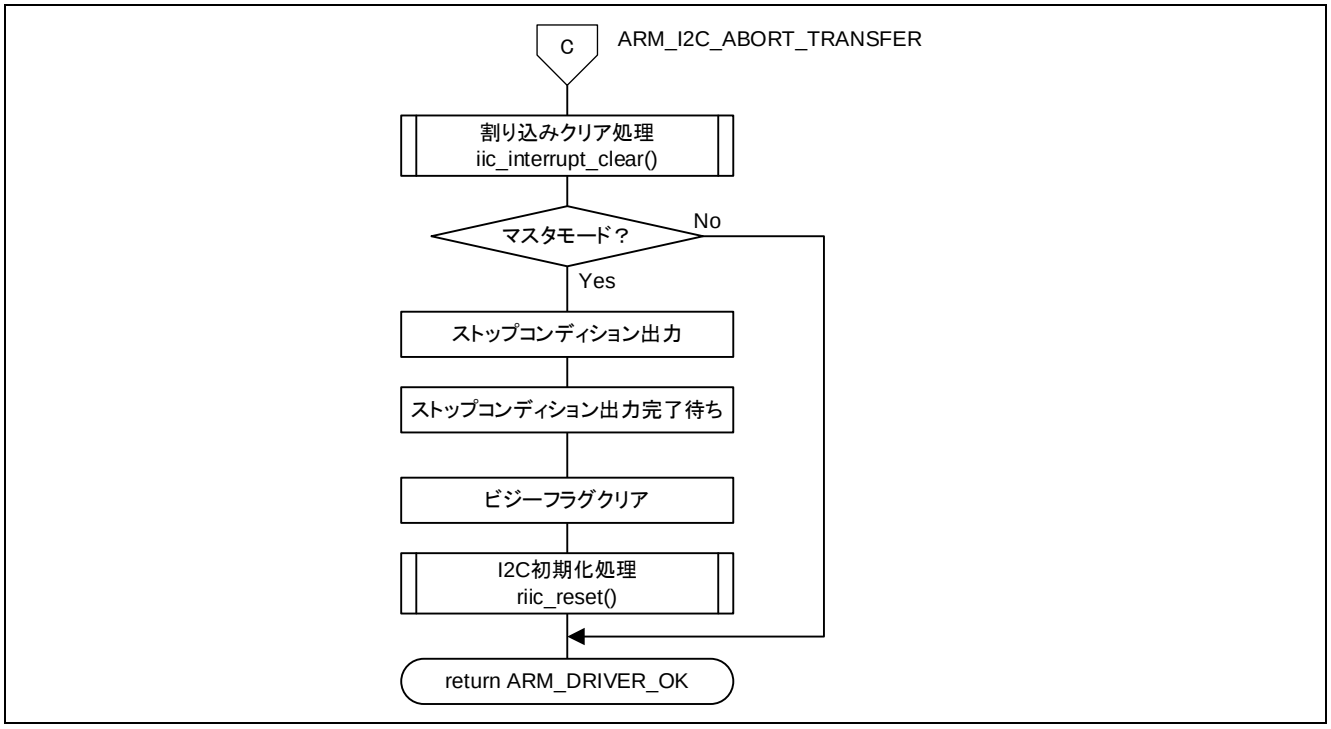

図 4-11 ARM_I2C_Control 関数処理フロー(3/3)

4.1.10　ARM_I2C_GetStatus 関数

表 4-11 ARM_I2C_GetStatus 関数仕様

書式	ARM_I2C_STATUS ARM_I2C_GetStatus(st_i2c_resources_t *p_i2c)
仕様説明	I2C のステータスを返します
引数	st_i2c_resources_t *p_i2c : I2C のリソース 対象の I2C 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 I2C 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]
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	
	main()
	{
	ARM_I2C_STATUS state;
	state = I2cDev0->GetStatus();
	
	}

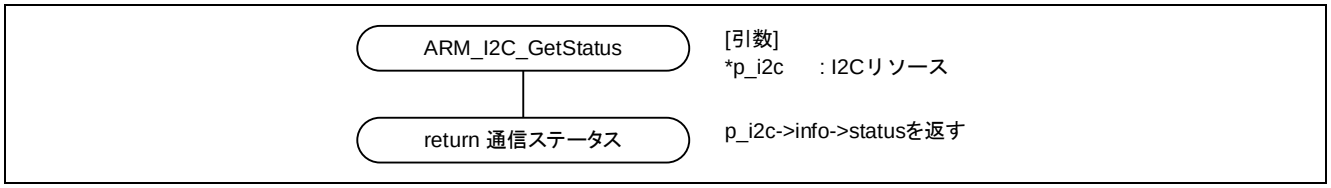

図 4-12 ARM_I2C_GetStatus 関数処理フロー

4.1.11　ARM_I2C_GetVersion 関数

表 4-12 ARM_I2C_GetVersion 関数仕様

書式	static ARM_DRIVER_VERSION ARM_I2C_GetVersion(void)
仕様説明	I2C ドライバのバージョンを取得します
引数	なし
戻り値	I2C ドライバのバージョン

備考	インスタンスからのアクセス時は I2C 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]
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	
	main()
	{
	　　ARM_DRIVER_VERSION version;
	　　version = I2cDev0->GetVersion();
	
	}

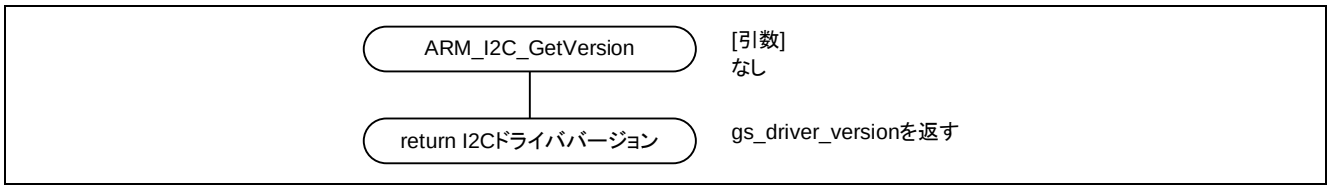

図 4-13 ARM_I2C_GetVersion 関数処理フロー

4.1.12　ARM_I2C_GetCapabilities 関数

表 4-13 ARM_I2C_ GetCapabilities 関数仕様

書式	static ARM_I2C_CAPABILITIES ARM_I2C_GetCapabilities(void)
仕様説明	I2C 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アクセス時は I2C 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]
	// I2C driver instance (RIIC0)
	extern ARM_DRIVER_I2C Driver_I2C0;
	ARM_DRIVER_I2C *i2cDev0 = &Driver_I2C0;
	
	main()
	{
	　　ARM_I2C_CAPABILITIES cap;
	　　cap = I2cDev0->GetCapabilities();
	
	}

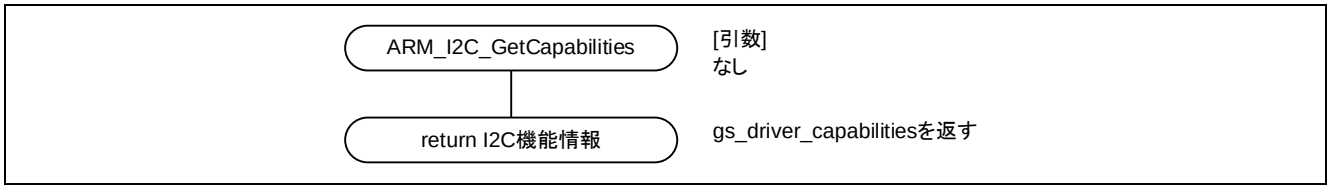

図 4-14 ARM_I2C_GetCapabilities 関数処理フロー

4.1.13　riic_bps_calc 関数

表 4-14 riic_bps_calc 関数仕様

書式	static int32_t riic_bps_calc (uint16_t kbps, st_i2c_reg_buf_t *reg_val)
仕様説明	バス速度の算出を行います
引数	uint16_t kbps : バス速度
	st_i2c_reg_buf_t *reg_val: レジスタ設定用バッファ バス速度の算出結果を格納するバッファ。
戻り値	ARM_DRIVER_OK　　バス速度算出成功
	ARM_DRIVER_ERROR　　バス速度算出失敗
備考	r_i2c_cfg.h の RIIC_BUS_SPEED_CAL_ENABLE により処理内容が異なります。

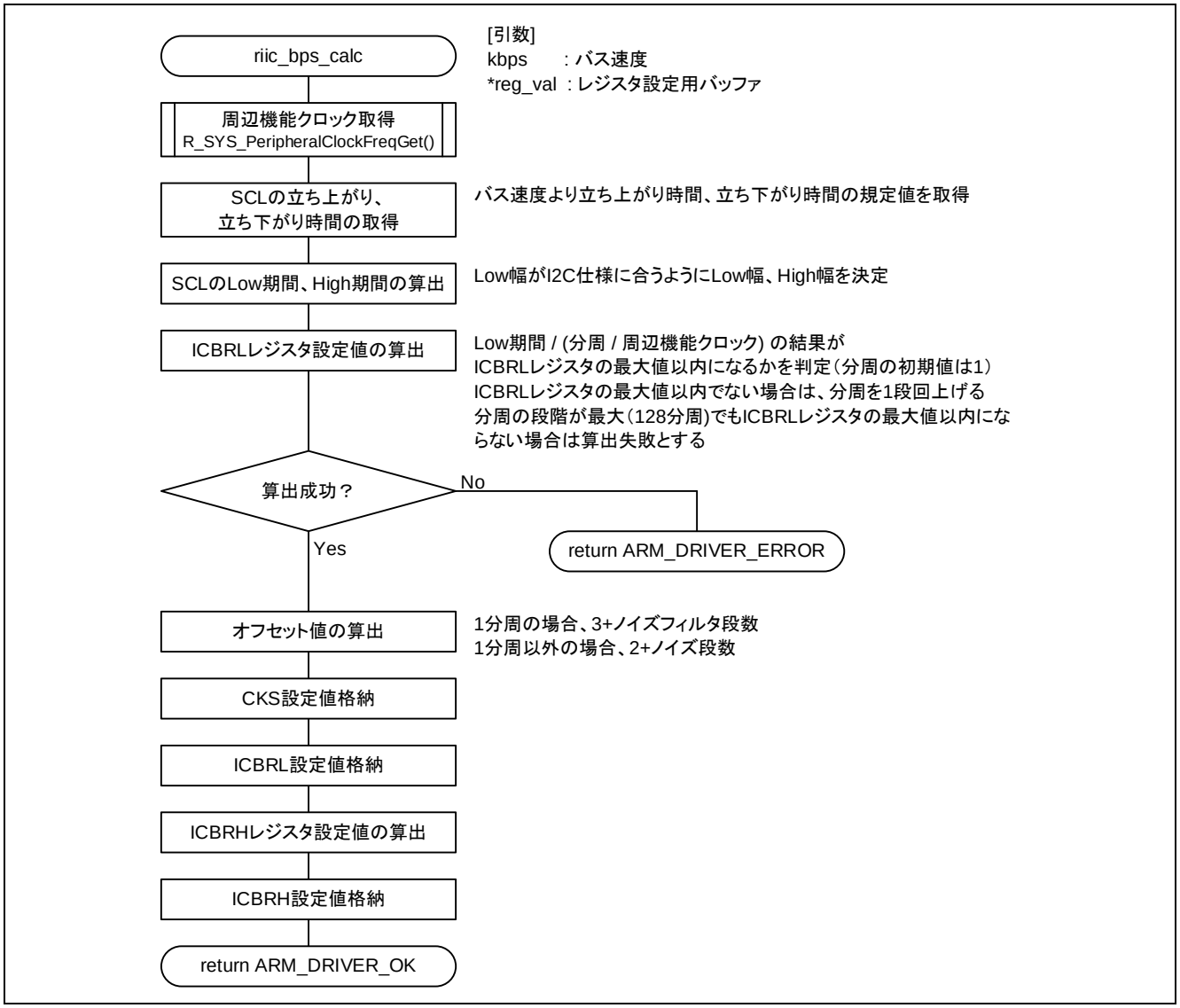

図 4-15 riic_bps_calc 関数処理フロー(RIIC_BUS_SPEED_CAL_ENABLE = 1 の場合)

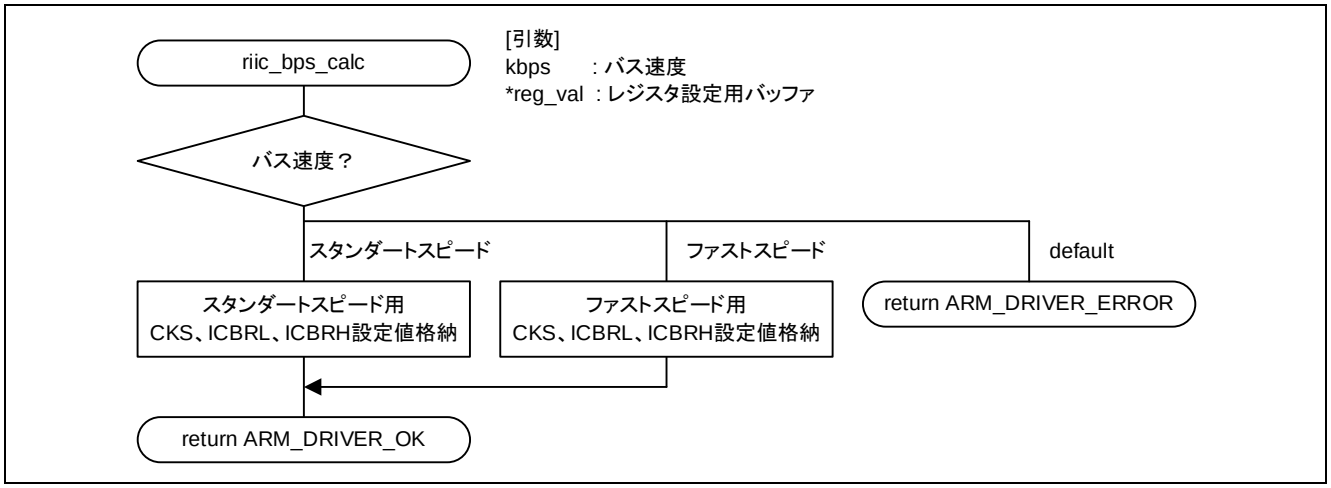

図 4-16 riic_bps_calc 関数処理フロー(RIIC_BUS_SPEED_CAL_ENABLE = 0 の場合)

4.1.14　riic_setting 関数

表 4-15 riic_setting 関数仕様

書式	static int32_t riic_setting(uint16_t bps, st_i2c_resources_t *p_i2c)	
仕様説明	RIIC のレジスタ設定を行います	
引数	uint16_t bps : バス速度	
	st_i2c_resources_t *p_i2c : I2C のリソース 対象の I2C のリソースを指定します。	
戻り値	ARM_DRIVER_OK	レジスタ設定成功
	ARM_DRIVER_ERROR	レジスタ設定失敗
備考	―	

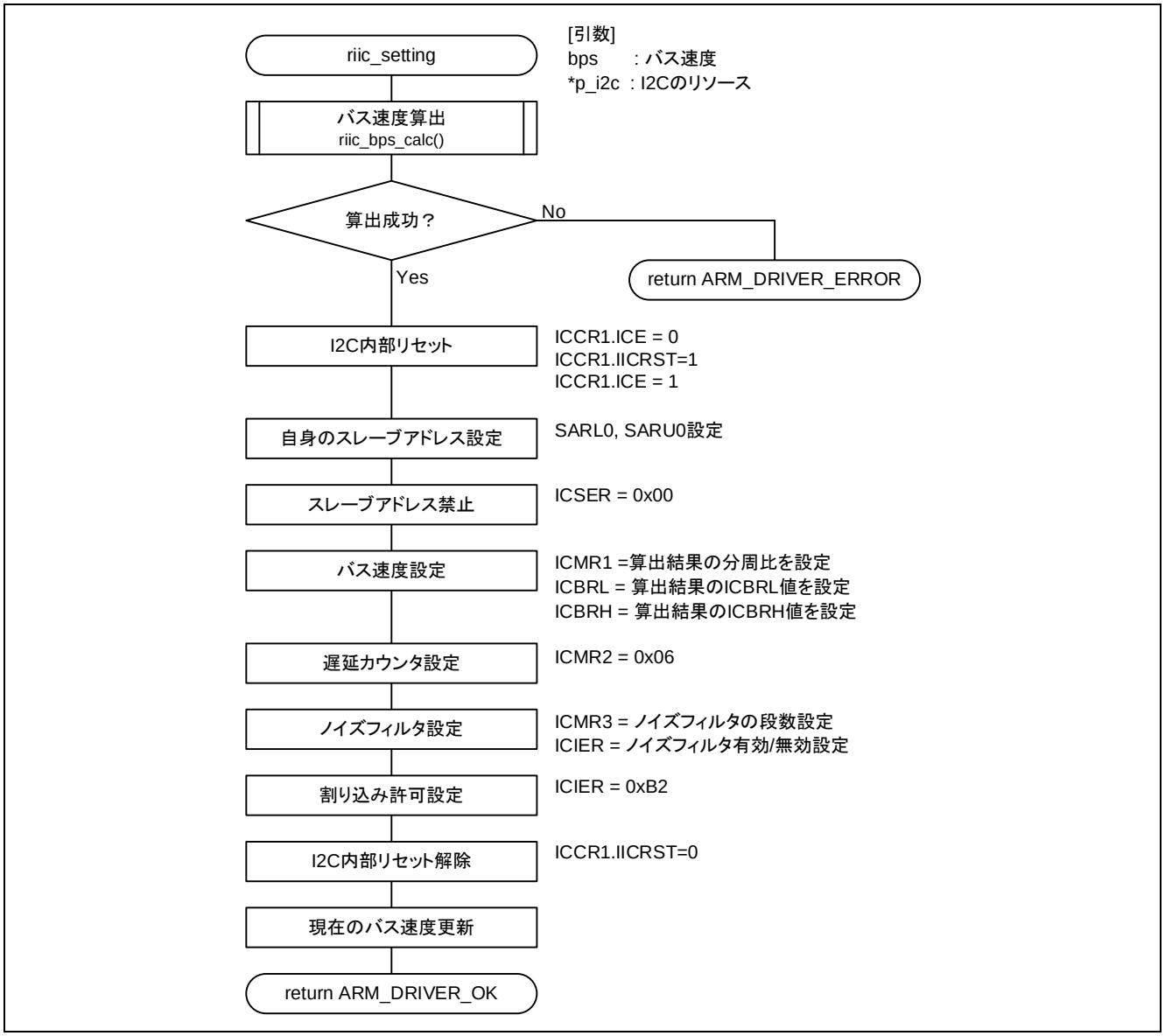

図 4-17 riic_setting 関数処理フロー

4.1.15　riic_reset 関数

表 4-16 riic_reset 関数仕様

書式	static void riic_reset(st_i2c_resources_t *p_i2c, bool en_recv)
仕様説明	RIIC の初期化処理を行います
引数	st_i2c_resources_t *p_i2c : I2C のリソース 対象の I2C のリソースを指定します。
	bool en_recv RIIC_RECV_ENABLE : 受信許可状態で初期化 RIIC_RECV_DISABLE : 受信禁止状態で初期化
戻り値	なし
備考	―

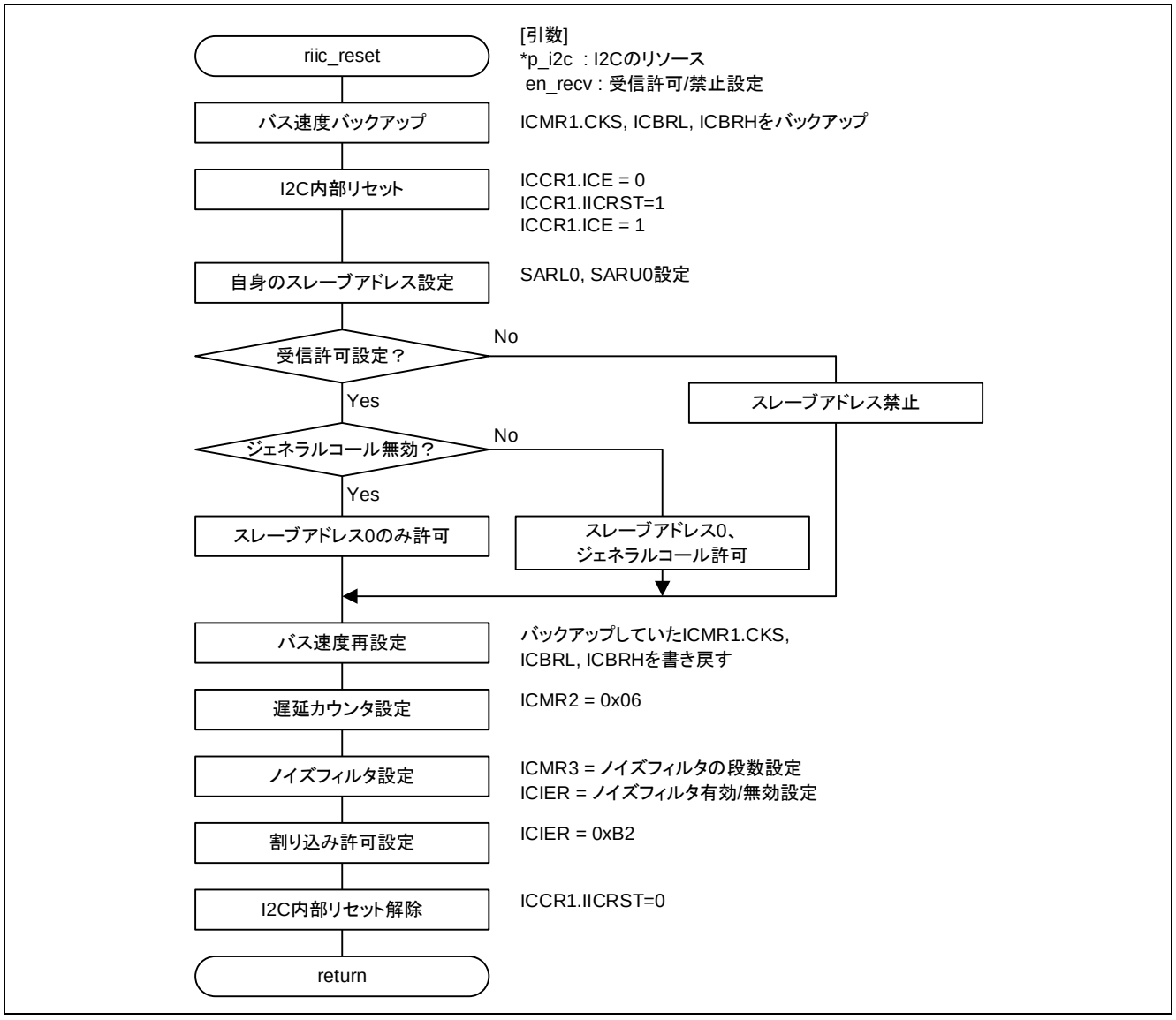

図 4-18 riic_reset 関数処理フロー

4.1.16　iic_txi_interrupt 関数

表 4-17 iic_txi_interrupt 関数仕様

書式	static void iic_txi_interrupt(st_i2c_resources_t *p_i2c)
仕様説明	TXI 割り込み処理
引数	st_i2c_resources_t *p_i2c : I2C のリソース 対象の I2C のリソースを指定します。
戻り値	なし
備考	―

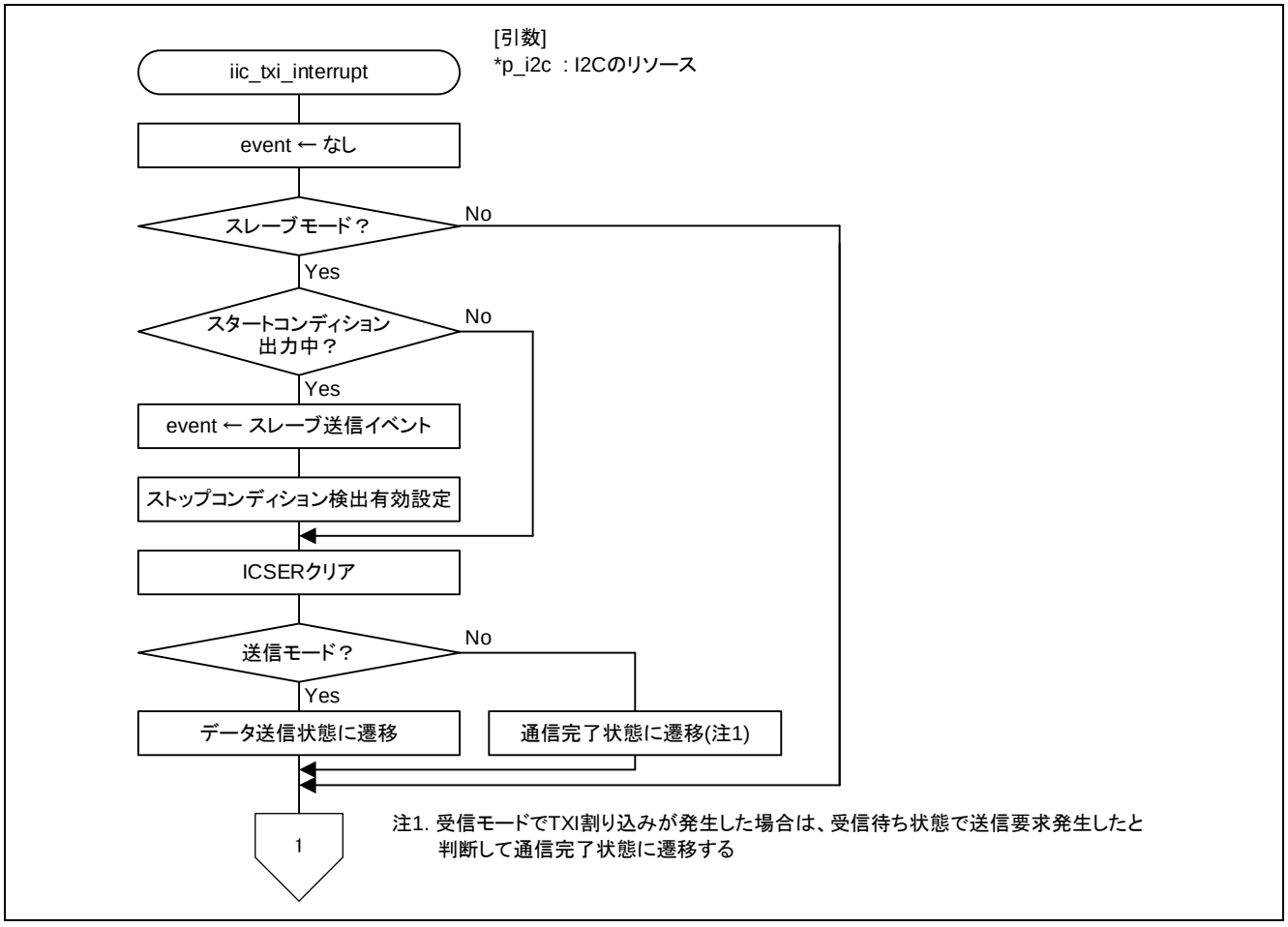

図 4-19 iic_txi_interrupt 関数処理フロー(1/2)

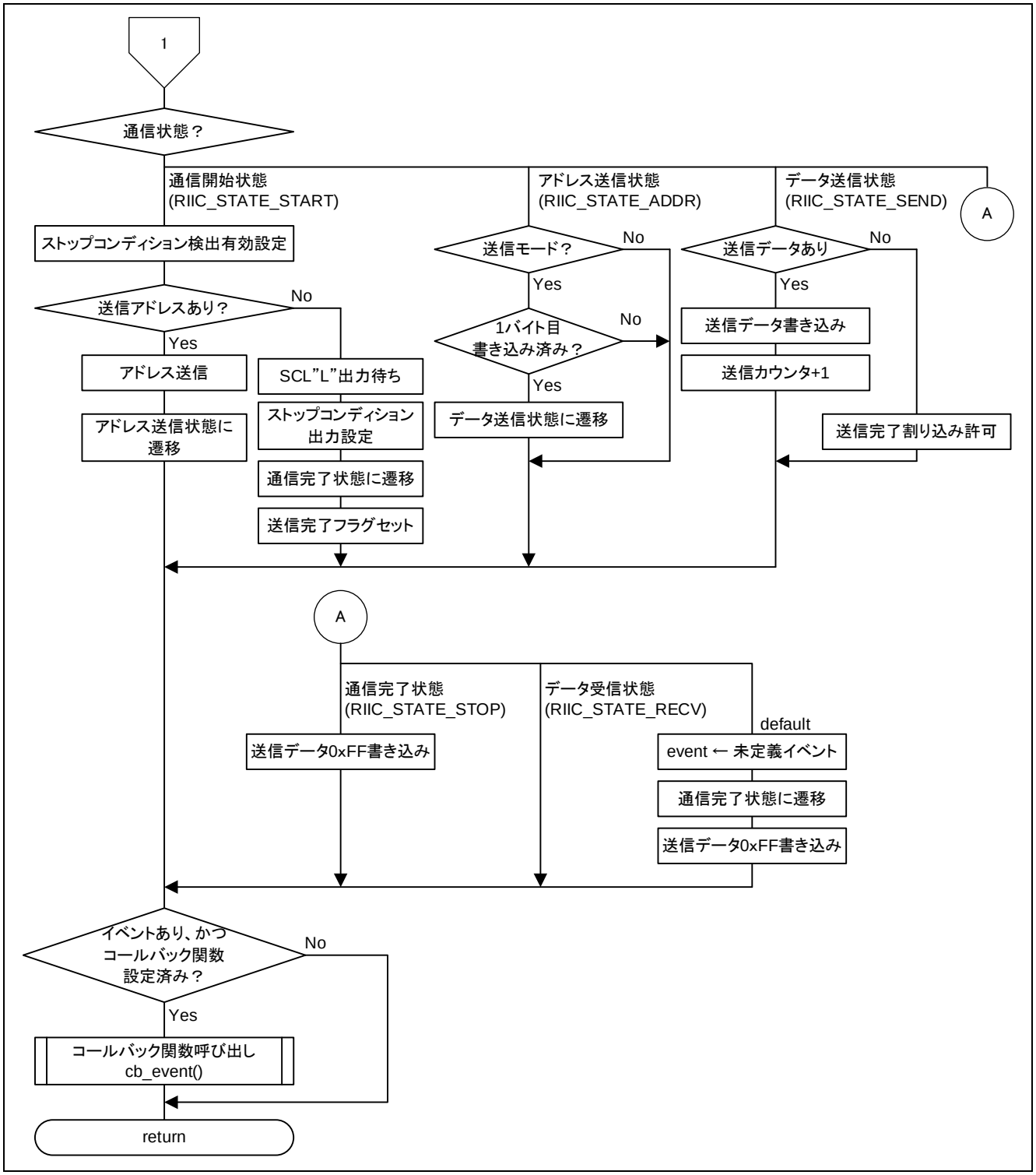

図　4-20 iic_txi_interrupt 関数処理フロー(2/2)

4.1.17　iic_tei_interrupt 関数

表 4-18 iic_tei_interrupt 関数仕様

書式	static void iic_tei_interrupt(st_i2c_resources_t *p_i2c)
仕様説明	TEI 割り込み処理
引数	st_i2c_resources_t *p_i2c : I2C のリソース 対象の I2C のリソースを指定します。
戻り値	なし
備考	―

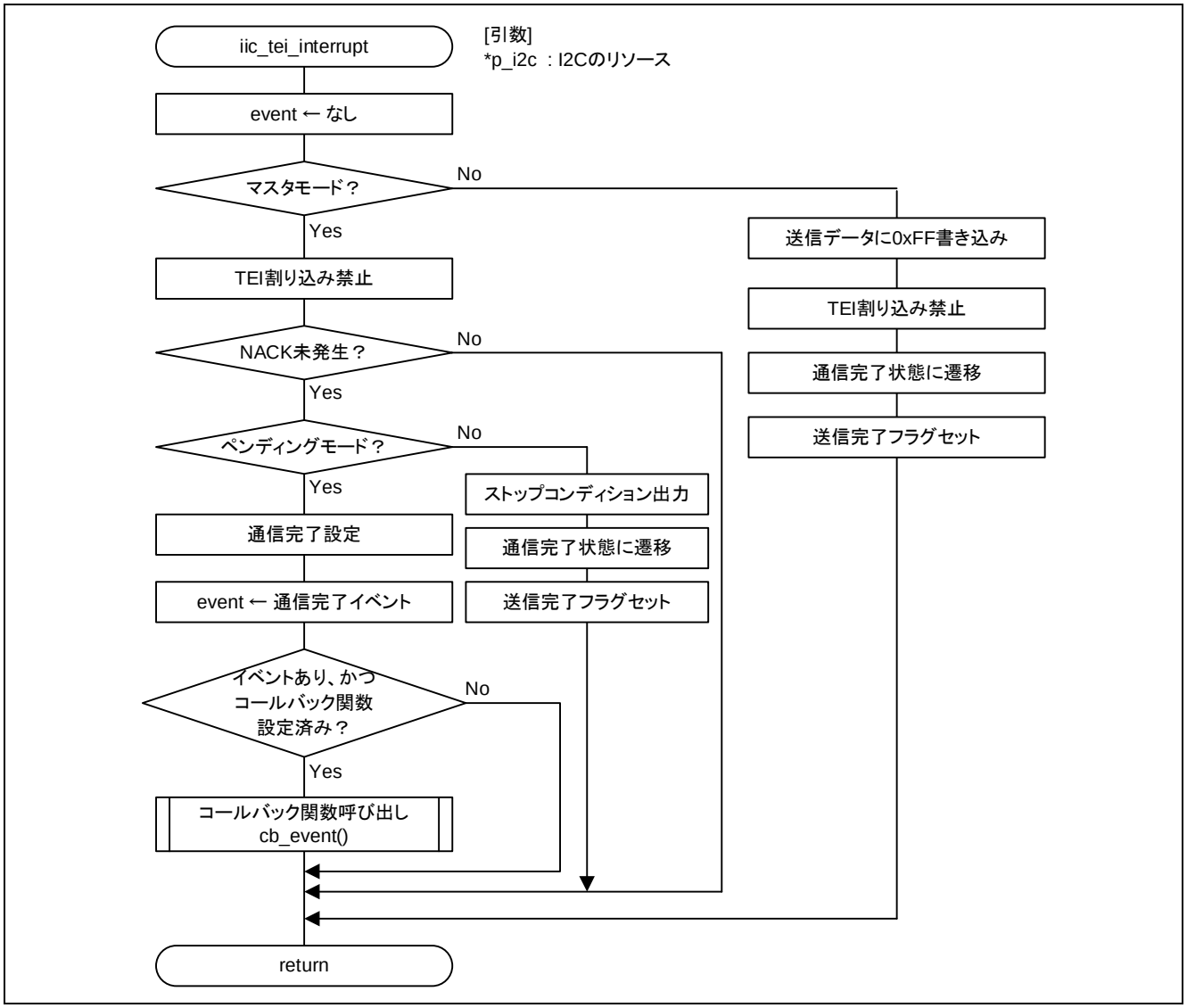

図 4-21 iic_tei_interrupt 関数処理フロー

4.1.18　iic_rxi_interrupt 関数

表 4-19 iic_rxi_interrupt 関数仕様

書式	static void iic_rxi_interrupt(st_i2c_resources_t *p_i2c)
仕様説明	RXI 割り込み処理
引数	st_i2c_resources_t *p_i2c : I2C のリソース 対象の I2C のリソースを指定します。
戻り値	なし
備考	―

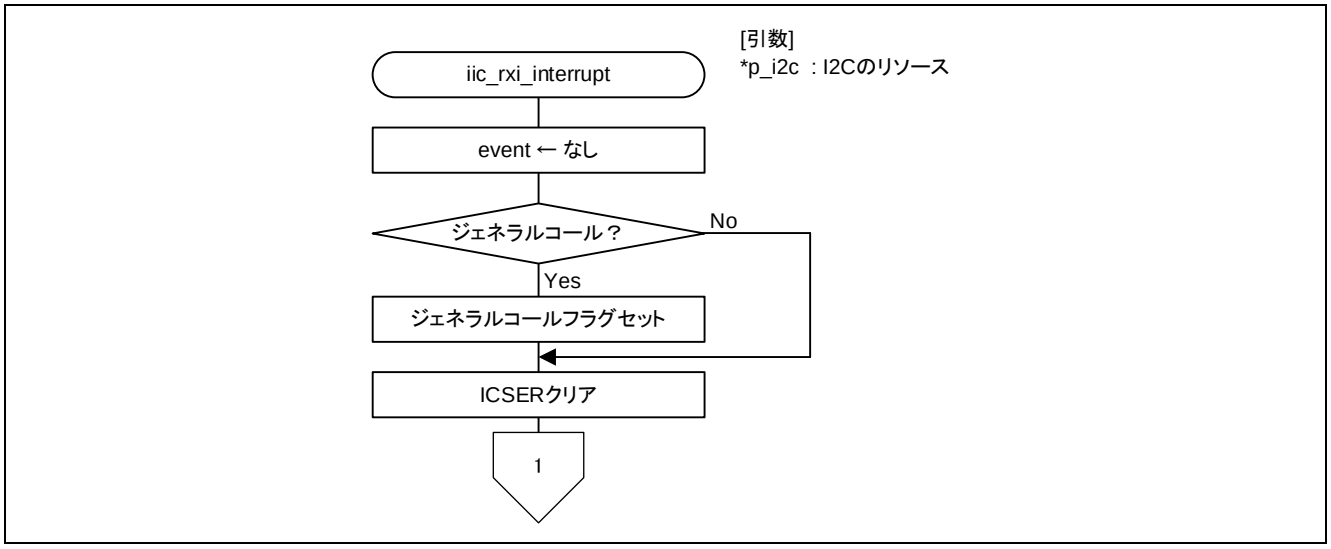

図 4-22 iic_rxi_interrupt 関数処理フロー(1/3)

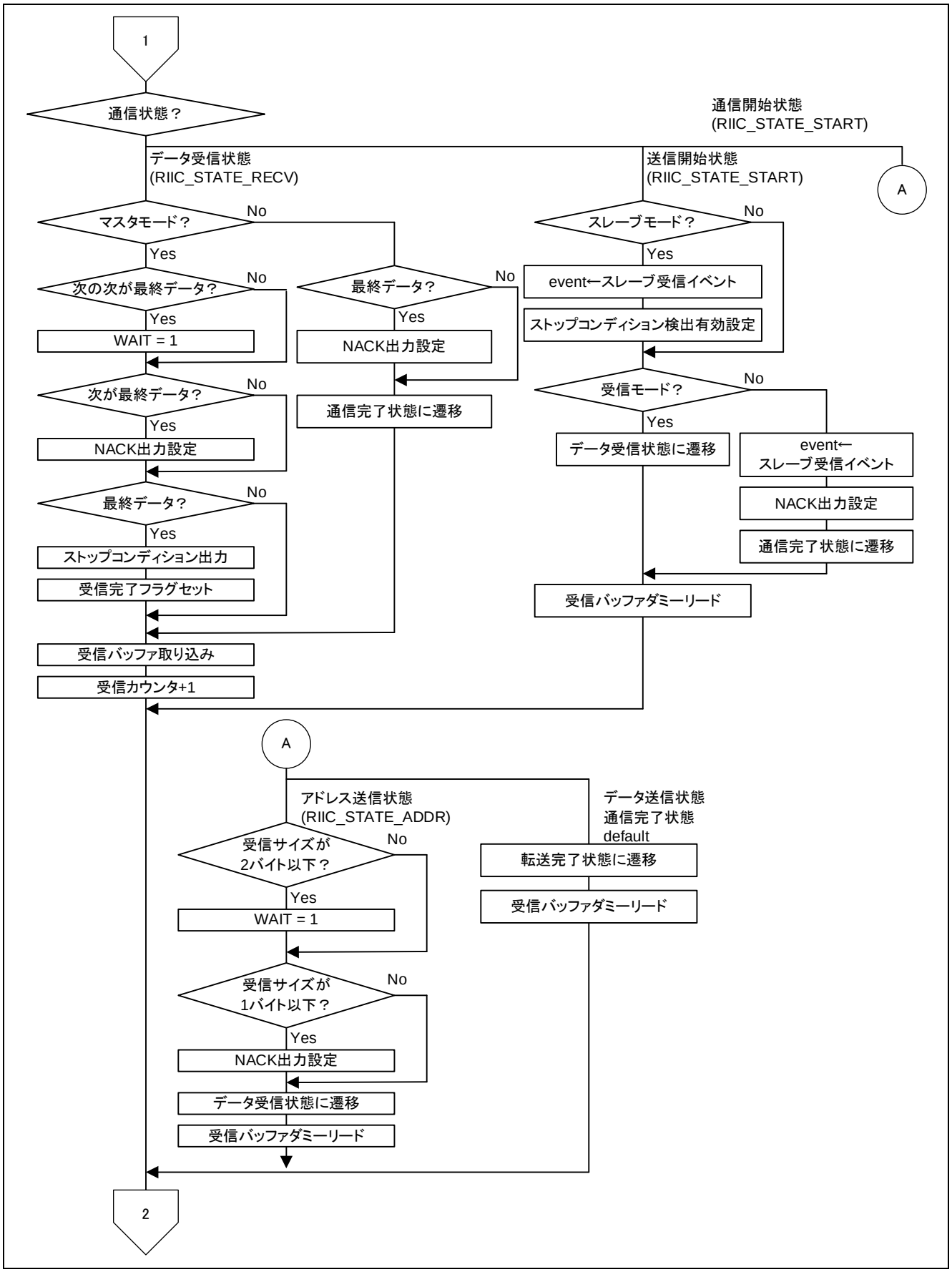

図 4-23 iic_rxi_interrupt 関数処理フロー(2/3)

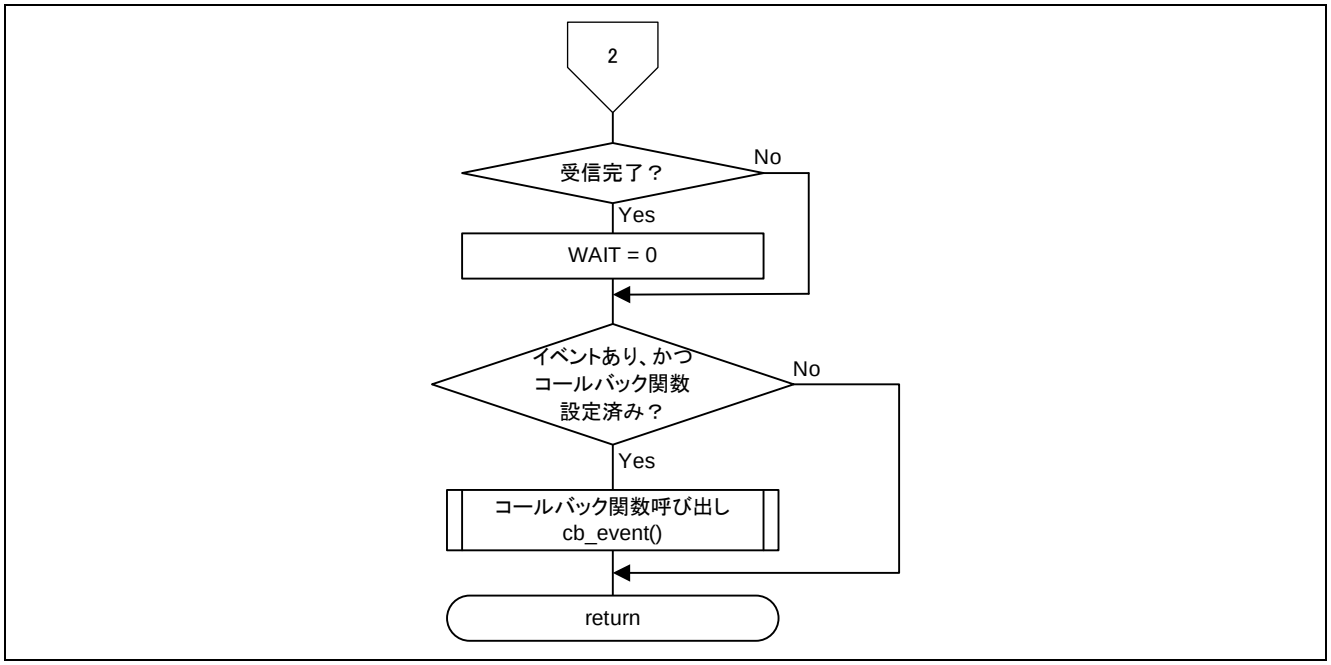

図 4-24 iic_rxi_interrupt 関数処理フロー(3/3)

4.1.19　iic_eei_interrupt 関数

表 4-20 iic_eei_interrupt 関数仕様

書式	static void iic_eei_interrupt(st_i2c_resources_t *p_i2c)
仕様説明	EEI 割り込み処理
引数	st_i2c_resources_t *p_i2c : I2C のリソース 対象の I2C のリソースを指定します。
戻り値	なし
備考	―

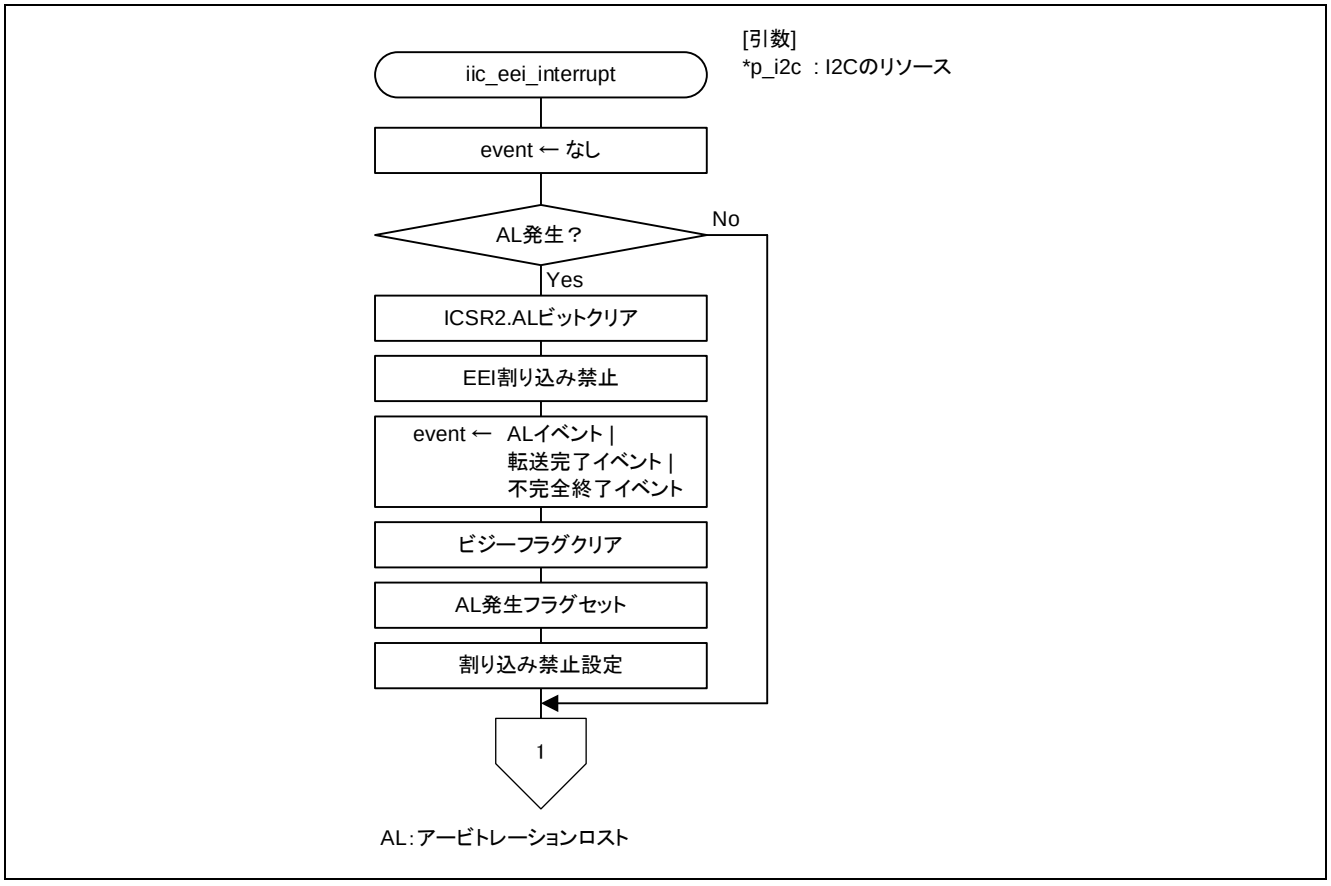

図 4-25 iic_eei_interrupt 関数処理フロー(1/3)

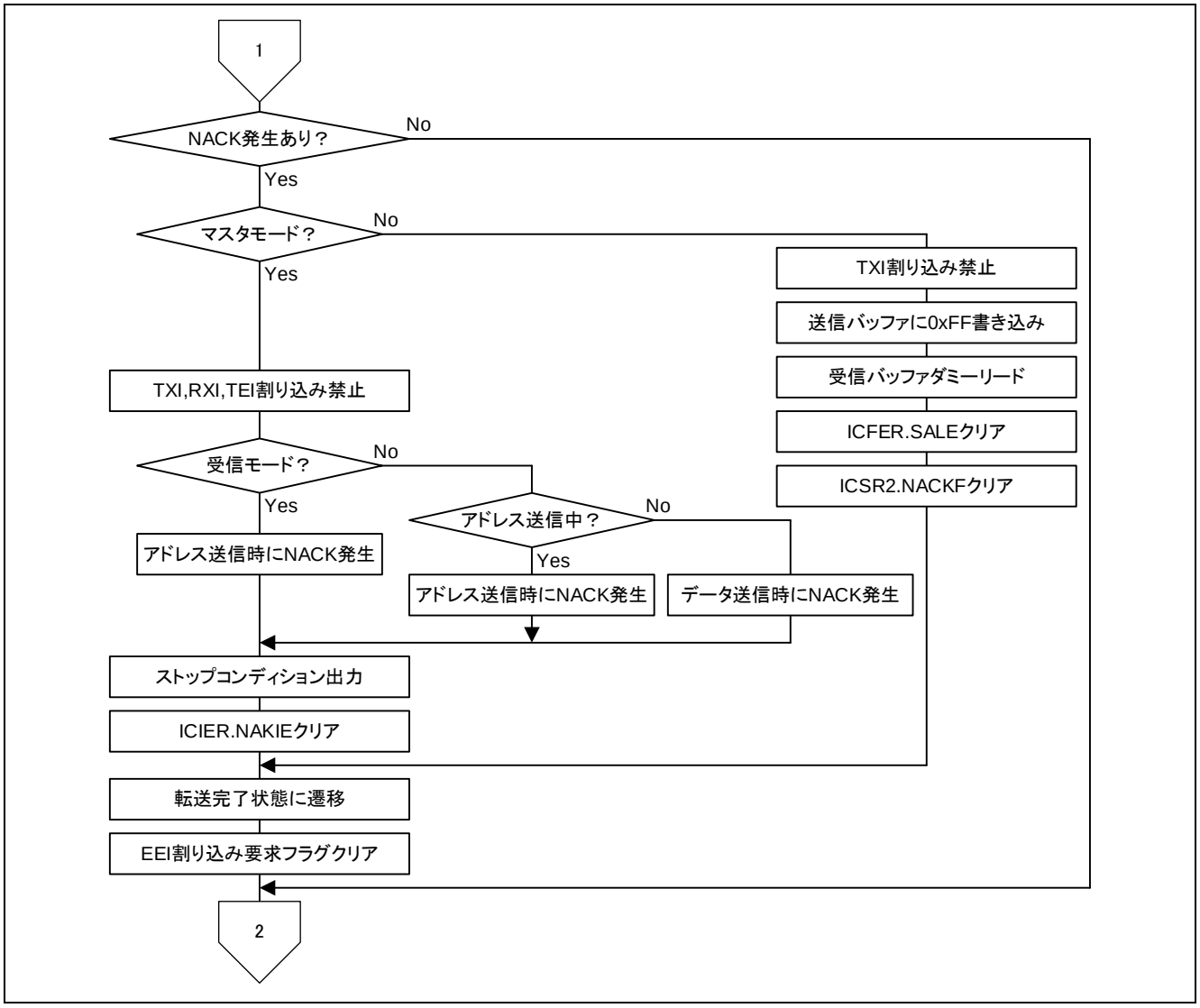

図 4-26 iic_eei_interrupt 関数処理フロー(2/3)

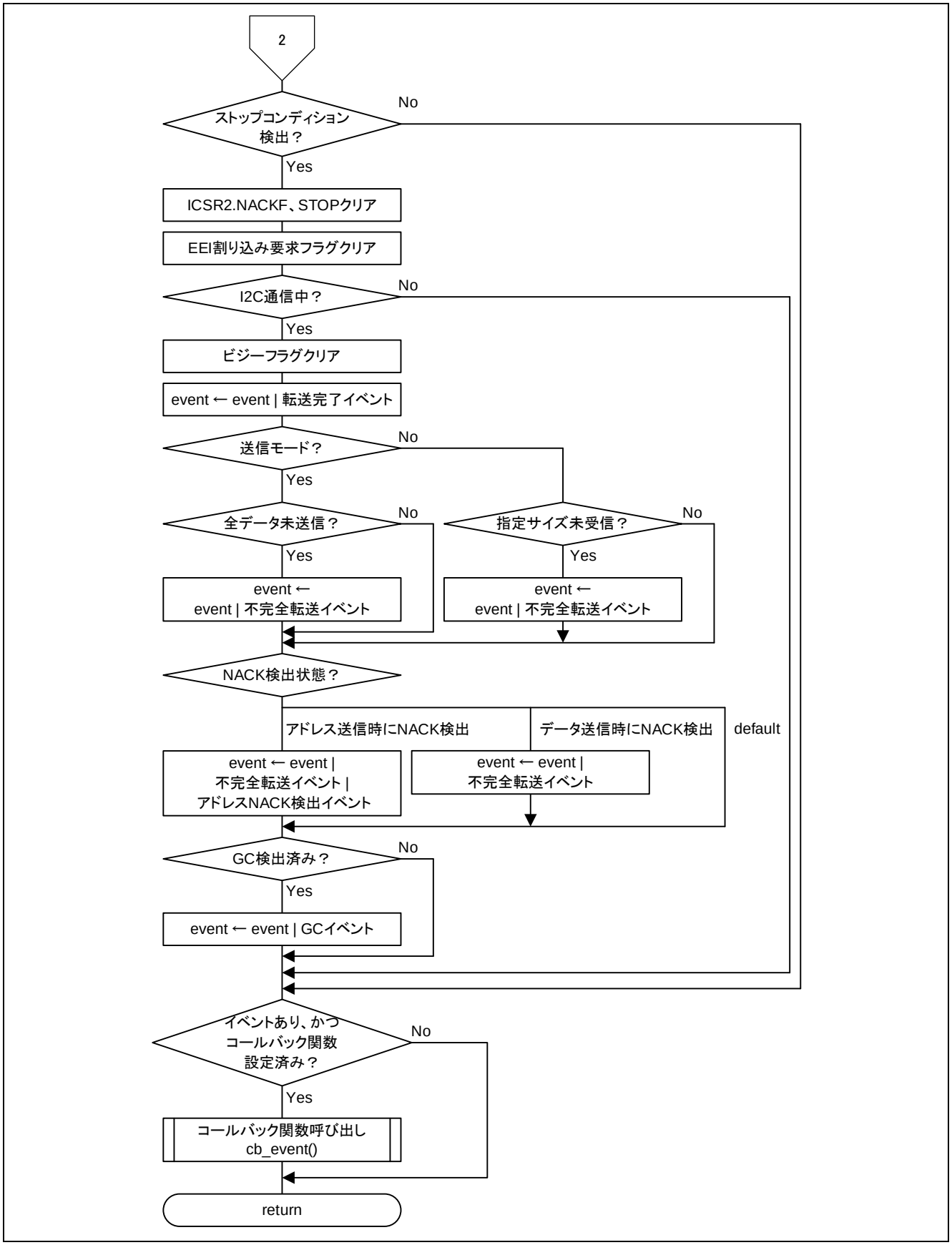

図 4-27 iic_eei_interrupt 関数処理フロー(3/3)

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 4-21 I2C マクロ定義一覧

定義	値	内容
R_RIIC0_ENABLE	(1)	RIIC0 リソース有効定義
R_RIIC1_ENABLE	(1)	RIIC1 リソース有効定義
RIIC_MODE_MASTER	(1)	マスタモード定義
RIIC_MODE_SLAVE	(0)	スレーブモード定義
RIIC_FLAG_INITIALIZED	(1U << 0)	初期化完了フラグ定義
RIIC_FLAG_POWERED	(1U << 1)	モジュール解除済みフラグ定義
RIIC_TRANSMITTER	(0)	送信動作中定義
RIIC_RECIVER	(1)	受信動作中定義
RIIC_MAX_DIV	((uint8_t)0x08)	分周判定テーブルの要素数
RIIC_STAD_SPPED_MAX	((uint16_t)100)	スタンダードスピードのバス速度
RIIC_FAST_SPPED_MAX	((uint16_t)400)	ファストスピードのバス速度
RIIC_ICBR_MAX	(32)	ICBR レジスタの最大値
RIIC_DEFAULT_ADDR	((uint32_t)0x00000000)	デフォルトのスレーブアドレス
RIIC_DEFAULT_BPS	(RIIC_STAD_SPPED_MAX)	デフォルトのバス速度
RIIC_WRITE	(0)	ライトモード定義
RIIC_READ	(1)	リードモード定義
RIIC_RECV_ENABLE	(1)	受信許可定義
RIIC_RECV_DISABLE	(0)	受信禁止定義
RIIC_ICMR3_DEF	RIIC_NOISE_FILTER == 0 のとき (0x00) RIIC_NOISE_FILTER が 0 以外 (0x00 | RIIC_NOISE_FILTER-1)	ICMR3 レジスタのデフォルト定義 （ノイズフィルタの有効/無効で値変更）
RIIC_ICFER_DEF	RIIC_NOISE_FILTER == 0 のとき (0x5A) RIIC_NOISE_FILTER が 0 以外 (0x7A)	ICFER レジスタのデフォルト定義 （ノイズフィルタの有効/無効で値変更）

4.2.2　e_i2c_driver_state_t 定義

ドライバの状態を示す定義です。

表 4-22 e_i2c_driver_state_t 定義一覧

定義	値	内容
RIIC_STATE_START	0	通信開始状態
RIIC_STATE_ADDR	1	アドレス送信状態
RIIC_STATE_SEND	2	データ送信状態
RIIC_STATE_RECV	3	データ受信状態
RIIC_STATE_STOP	4	通信完了状態

4.2.3　e_i2c_nack_state_t 定義

NACK 発生状態を示す定義です。

表 4-23 e_i2c_nack_state_t 定義一覧

定義	値	内容
RIIC_NACK_NONE	0	NACK 未発生
RIIC_NACK_ADDR	1	アドレス部で NACK 発生
RIIC_NACK_DATA	2	データ部で NACK 発生

4.3　構造体定義

4.3.1　st_i2c_resources_t 構造体

RIIC のリソースを構成する構造体です。

表 4-24 st_i2c_resources_t 構造体

要素名	型	内容
*reg	volatile IIC0_Type	対象の RIIC レジスタを示します
pin_set	r_pinset_t	端子設定用関数ポインタ
pin_clr	r_pinclr_t	端子解除用関数ポインタ
*info	st_i2c_info_t	I2C 状態情報
*xfer_info	st_i2c_transfer_info_t	転送情報
lock_id	e_system_mcu_lock_t	RIIC ロック ID
mstp_id	e_lpm_mstp_t	RIIC モジュールストップ ID
txi_irq	IRQn_Type	TXI 割り込みの NVIC 割り当て番号
tei_irq	IRQn_Type	TEI 割り込みの NVIC 割り当て番号
rxi_irq	IRQn_Type	RXI 割り込みの NVIC 割り当て番号
eei_irq	IRQn_Type	EEI 割り込みの NVIC 割り当て番号
txi_iesr_val	uint32_t	TXI 割り込み用 IESR レジスタ設定値
tei_iesr_val	uint32_t	TEI 割り込み用 IESR レジスタ設定値
rxi_iesr_val	uint32_t	RXI 割り込み用 IESR レジスタ設定値
eei_iesr_val	uint32_t	EEI 割り込み用 IESR レジスタ設定値
txi_priority	uint32_t	TXI 割り込み優先レベル
tei_priority	uint32_t	TEI 割り込み優先レベル
rxi_priority	uint32_t	RXI 割り込み優先レベル
eei_priority	uint32_t	EEI 割り込み優先レベル
txi_callback	system_int_cb_t	TXI 割り込みコールバック関数
tei_callback	system_int_cb_t	TEI 割り込みコールバック関数
rxi_callback	system_int_cb_t	RXI 割り込みコールバック関数
eei_callback	system_int_cb_t	EEI 割り込みコールバック関数

4.3.2　st_i2c_reg_buf_t 構造体

バス速度算出時など、一時的にレジスタ設定値を格納するバッファ用の構造体です。

表 4-25 st_i2c_reg_buf_t 構造体

要素名	型	内容
cks	uint8_t	ICMR1.CKS ビット設定値
icbrl	uint8_t	ICBRL レジスタ設定値
icbrh	uint8_t	ICBRH レジスタ設定値

4.3.3　st_i2c_info_t 構造体

RIIC の状態を管理するための構造体です。

表 4-26 st_i2c_info_t 構造体

要素名	型	内容
cb_event	ARM_I2C_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_I2C_STATUS	ステータスフラグ
flags	uint8_t	ドライバ設定フラグ 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 : RIIC 設定済み状態(0:未設定、1:設定済み)
own_sla	uint32_t	自身のスレーブアドレス
bps	uint16_t	現在のバス速度
state	e_i2c_driver_state_t	ドライバの状態 RIIC_STATE_START : RIIC 開始状態 RIIC_STATE_ADDR： アドレス出力中 RIIC_STATE_SEND： データ送信中 RIIC_STATE_RECV： データ受信中 RIIC_STATE_STOP： 通信完了
nack	e_i2c_nack_state_t	NACK 発生状態 RIIC_NACK_NONE : NACK 未発生 RIIC_NACK_ADDR： アドレス部で NACK 発生 RIIC_NACK_DATA： データ部で NACK 発生
pending	bool	ペンディングモード設定 0： 通常モードで動作（ストップコンディション出力） 1： ペンディングモードで動作 （ストップコンディションを出力しない）

4.3.4　st_i2c_transfer_info_t 構造体

転送情報を管理するための構造体です。

表 4-27 st_i2c_transfer_info_t 構造体

要素名	型	内容
sla	uint32_t	転送先のアドレス
rx_num	uint32_t	受信サイズ
tx_num	uint32_t	送信サイズ
*rx_buf	uint8_t	受信データ格納バッファのポインタ
*tx_buf	const uint8_t	送信データ格納バッファのポインタ
cnt	uint32_t	現在の送信/受信サイズ
f_tx_end	uint8_t	送信完了フラグ

4.4　外部関数の呼び出し

I2C ドライバ API から呼び出される外部関数を示します。

表 4-28 I2C 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 1）
Initialize	R_SYS_IrqEventLinkSet	なし
	R_NVIC_SetPriority	なし
	R_NVIC_GetPriority	なし
	R_SYS_ResourceLock	なし
	R_RIIC_Pinset_CHn (n=0,1)	なし
Uninitialize	R_RIIC_Pinclr_CHn (n=0,1)	なし
	R_SYS_ResourceUnlock	なし
	R_NVIC_DisableIRQ	モジュールストップ解除状態で Uninitialize 関数実行時（PowerControl(ARM_POWER_FULL)実行後に Uninitialize 実行）
	R_LPM_ModuleStop	
PowerControl	R_NVIC_DisableIRQ	なし
	R_LPM_ModuleStop	ARM_POWER_OFF 指定時（モジュールストップ遷移） または ARM_POWER_FULL 指定時かつ初期化失敗時
	R_LPM_ModuleStart	ARM_POWER_FULL 指定時（モジュールストップ解除）
	R_SYS_PeripheralClockFreqGet	ARM_POWER_FULL 指定時（モジュールストップ解除）かつ RIIC_BUS_SPEED_CAL_ENABLE に”1”設定時（バス速度自動計算有効）
MasterTransmit	R_NVIC_DisableIRQ	なし
	R_SYS_IrqStatusClear	なし
	R_NVIC_ClearPendingIRQ	なし
	R_NVIC_EnableIRQ	なし
MasterReceive	R_NVIC_DisableIRQ	なし
	R_SYS_IrqStatusClear	なし
	R_NVIC_ClearPendingIRQ	なし
	R_NVIC_EnableIRQ	なし
SlaveTransmit	R_NVIC_DisableIRQ	なし
	R_SYS_IrqStatusClear	なし
	R_NVIC_ClearPendingIRQ	なし
	R_NVIC_EnableIRQ	なし
SlaveReceive	R_NVIC_DisableIRQ	なし
	R_SYS_IrqStatusClear	なし
	R_NVIC_ClearPendingIRQ	なし
	R_NVIC_EnableIRQ	なし
GetDataCount	-	-

注1. 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 4-29 I2C 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
Control	R_SYS_PeripheralClockFreqGet	ARM_I2C_BUS_SPEED コマンド実行時、かつ RIIC_BUS_SPEED_CAL_ENABLE に”1”設定時（バス速度自動計算有効）
	R_SYS_SoftwareDelay	以下のコマンド実行時 ・ARM_I2C_BUS_CLEAR ・ARM_I2C_ABORT_TRANSFER
	R_NVIC_DisableIRQ	ARM_I2C_ABORT_TRANSFER コマンド実行時
	R_SYS_IrqStatusClear	
	R_NVIC_ClearPendingIRQ	
	R_NVIC_EnableIRQ	
GetStatus	-	-
GetVersion	-	-
GetCapabilities	-	-

5.　使用上の注意

5.1　　NVIC への I2C 割り込み登録

I2C ドライバを使用する場合は、r_system_cfg.h にて NVIC に受信データフル割り込み(RXI)、送信終了割り込み(TEI)、送信データエンプティ割り込み(TXI)、通信エラー/イベント発生割り込み(EEI)を登録してください。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」を参照ください。

NVIC に I2C 割り込みが登録されていない場合、ARM_I2C_Initialize 関数実行時に ARM_DRIVER_ERROR が戻ります。

```
・・・

#define SYSTEM_CFG_EVENT_NUMBER_ADC140_WCMPM
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 0/4/8/12/16/20/24/28 only */

#define SYSTEM_CFG_EVENT_NUMBER_IIC0_RXI
    (SYSTEM_IRQ_EVENT_NUMBER0)  /*!< Numbers 0/4/8/12/16/20/24/28 only */

#define SYSTEM_CFG_EVENT_NUMBER_CCC_PRD
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 0/4/8/12/16/20/24/28 only */

・・・

#define SYSTEM_CFG_EVENT_NUMBER_ADC140_WCMPUM
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 1/5/9/13/17/21/25/29 only */

#define SYSTEM_CFG_EVENT_NUMBER_IIC0_TXI
    (SYSTEM_IRQ_EVENT_NUMBER1)  /*!< Numbers 1/5/9/13/17/21/25/29 only */

#define SYSTEM_CFG_EVENT_NUMBER_DOC_DOPCI
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 1/5/9/13/17/21/25/29 only */

・・・

#define SYSTEM_CFG_EVENT_NUMBER_ADC140_GCADI
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 2/6/10/14/18/22/26/30 only */

#define SYSTEM_CFG_EVENT_NUMBER_IIC0_TEI
    (SYSTEM_IRQ_EVENT_NUMBER2)  /*!< Numbers 2/6/10/14/18/22/26/30 only */

#define SYSTEM_CFG_EVENT_NUMBER_CAC_MENDI
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 2/6/10/14/18/22/26/30 only */

・・・

#define SYSTEM_CFG_EVENT_NUMBER_ACMP_CMPI
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 3/7/11/15/19/23/27/31 only */

#define SYSTEM_CFG_EVENT_NUMBER_IIC0_EEI
    (SYSTEM_IRQ_EVENT_NUMBER3)  /*!< Numbers 3/7/11/15/19/23/27/31 only */

#define SYSTEM_CFG_EVENT_NUMBER_CAC_OVFI
    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 3/7/11/15/19/23/27/31 only */
```

図 5-1 r_system_cfg.h での NVIC への割り込み登録例（RIIC0 を使用）

5.2　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。

5.3　端子設定について

本ドライバで使用する端子は、pin.c にて設定する必要があります。端子設定の詳細については「2.3 端子設定」を参照してください。

5.4　バス速度自動計算有効時の注意事項

バス速度の自動計算有効時（RIIC_BUS_SPEED_CAL_ENABLE = 1）、バス速度を設定した時点での PCLKB から、I2C バス仕様の規定を満たす値を算出して、レジスタの設定を行います。算出時には SCL ラインの立ち上がり時間、立ち下がり時間も考慮に入れて算出しています。そのため、PCLKB を変更した場合は、再度バス速度の設定を行ってください。

バス速度設定時に使用する I2C バス仕様の SCL クロック規定値と SCL ラインの立ち上がり、立ち下がり時間を表 5-1 に示します。

表 5-1 バス速度設定時の規定値

バス速度	項目	値
スタンダードモード（100kbps）	SCL クロックの Low 期間規定値	4.7us(min)
	SCL クロックの High 期間規定値	4.0us(min)
	SCL ラインの立ち上がり時間	RIIC_STAD_SCL_UP_TIME で定義 (初期値:1000ns)
	SCL ラインの立ち下がり時間	RIIC_STAD_SCL_DOWN_TIME で定義 (初期値:300ns)
ファストモード（400kbps）	SCL クロックの Low 期間規定値	1.3us(min)
	SCL クロックの High 期間規定値	0.6us(min)
	SCL ラインの立ち上がり時間	RIIC_FAST_SCL_UP_TIME で定義 (初期値:300ns)
	SCL ラインの立ち下がり時間	RIIC_FAST_SCL_DOWN_TIME で定義 (初期値:300ns)

算出結果は規定を満たす値を算出しますが、PCLKB 周波数が遅い場合は、バス速度の誤差が大きくなる可能性があります。

表 5-2 自動計算によるレジスタ設定結果例と誤差（ノイズフィルタ 2 段指定時）

バス速度	動作周波数 PCLKB	レジスタ設定値			出力期待値	
		ICMR1. CKS[2:0]	ICBRH. BRH[4:0]	ICBRL. BRL[4:0]	転送速度	誤差
100kbps	32MHz	011b	12(0Ch)	15(0FH)	99.5kbps	0.5%
	20MHz	010b	16(10h)	20(14h)	99.0kbps	1.0%
	8MHz	001b	12(0Ch)	15(0Fh)	99.5kbps	0.5%
	2MHz	000b	3(03h)	1(01H)	120.5kbps	20.5%
	1MHz	000b	0(00h)	0(00h)	88.5kbps	11.5%
400kbps	32MHz	001b	6(06h)	17(11h)	394.1kbps	1.5%
	20MHz	000b	7(07h)	21(15h)	400.0kbps	0.0%
	8MHz	000b	0(00h)	5(05h)	404.0kbps	1.0%
	2MHz	000b	0(00h)	0(00h)	178.6kbps	55.4%
	1MHz	000b	0(00h)	0(00h)	94.3kbps	76.4%

注1. 出力期待値のバス速度は以下の計算式で算出しています。
tr : SCL ラインの立ち上がり時間、tf：SCL ラインの立ち下がり時間、nf：ノイズフィルタの段数
スタンダードスピード時：tr = 1000ns、tf = 300ns
ファストスピード時：tr = 300ns、tf = 300ns
1) CKS[2:0] = 000b の場合 ：
転送速度 = 1/{[(BRH + 3 + nf) + (BRL + 3 + nf)]/(PCLKB) + tr + tf}
2) CKS[2:0] ≠ 000b の場合 ：
転送速度 = 1/{[(BRH + 2 + nf) + (BRL + 2 + nf)]/ (PCLKB /分周比) + tr + tf}

表 5-3 自動計算によるレジスタ設定結果例と誤差（ノイズフィルタなし指定時）

バス速度	動作周波数 PCLKB	レジスタ設定値			出力期待値	
		ICMR1. CKS[2:0]	ICBRH. BRH[4:0]	ICBRL. BRL[4:0]	転送速度	誤差
100kbps	32MHz	011b	14(0Eh)	17(11H)	99.5kbps	0.5%
	20MHz	010b	18(12h)	22(16h)	99.0kbps	1.0%
	8MHz	001b	14(0Eh)	17(11h)	99.5kbps	0.5%
	2MHz	000b	5(05h)	6(06H)	102.0kbps	2.0%
	1MHz	000b	1(01h)	2(02h)	97.1kbps	2.9%
400kbps	32MHz	001b	7(07h)	18(12h)	414.5kbps	3.6%
	20MHz	000b	9(09h)	23(17h)	400.0kbps	0.0%
	8MHz	000b	2(02h)	7(07h)	404.0kbps	1.0%
	2MHz	000b	0(00h)	0(00h)	277.8kbps	30.6%
	1MHz	000b	0(00h)	0(00h)	151.5kbps	62.1%

注1. 出力期待値のバス速度は以下の計算式で算出しています。
tr : SCL ラインの立ち上がり時間、tf：SCL ラインの立ち下がり時間
スタンダードスピード時：tr = 1000ns、tf = 300ns
ファストスピード時：tr = 300ns、tf = 300ns
1) CKS[2:0] = 000b の場合 ：
転送速度 = 1/{[(BRH + 3) + (BRL + 3)]/ PCLKB + tr + tf}
2) CKS[2:0] ≠ 000b の場合 ：
転送速度 = 1/{[(BRH + 2) + (BRL + 2)]/ (PCLKB /分周比) + tr + tf}

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.10.2019	ー	初版
1.01	Oct.28.2019	27,66 プログラム 10,74	定義名「RIIC_NOIZE_FILTER」を「RIIC_NOISE_FILTER」に変更
			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
1.02	Dec.16.2019	ー	256KB グループに対応
1.03	Feb.21.2020	プログラム (256KB, 1500KB)	ICMR3.ACKBT 設定手順を修正
1.04	Nov.05.2020	ー	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。