RX ファミリ

IrDA モジュール Firmware Integration Technology

R01AN2175JJ0102
Rev.1.02
2024.11.15

要旨

本アプリケーションノートは、Firmware Integration Technology (FIT)を使用した IrDA モジュールについて説明します。

本モジュールは、Infrared Data Association インタフェース(IrDA)とシリアルコミュニケーションインタフェース(SCI)を使用して IrDA 通信波形を生成し、赤外線によるデータ送受信を行います。以降、本モジュールを IrDA FIT モジュールと称します。

対象デバイス

本モジュールは、現時点で次のデバイスでサポートされています。

・RX113 グループ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

関連アプリケーションノート

本アプリケーションノートに関連するアプリケーションノートを以下に示します。併せて参照してください。

Firmware Integration Technology ユーザーズマニュアル(R01AN1833JU)
ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685JU)
e^{2}studio に組み込む方法 Firmware Integration Technology (R01AN1723JU)
CS+に組み込む方法 Firmware Integration Technology (R01AN1826JJ)

目次

1. 概要

本モジュールは、RX シリーズの周辺機能である IrDA をサポートし、IrDA 規格バージョン 1.0 に基づく IrDA 通信波形の送受信を行います。

本モジュールは TXI、TEI、RXI、および ERI 割り込みをサポートしています。

本モジュールは、DMAC、DTC、ELC と組み合わせて使用することや、多重割り込みには対応していません。また、本モジュールでは割り込みを使用するため、I フラグを“1”にしてください。

1.1 IrDA FIT モジュールとは

IrDA FIT モジュールは API として、プロジェクトに組み込んで使用します。モジュールの組み込み方については、2 章の FIT モジュールの追加方法 を参照してください。

1.2 API の概要

表 1.1 に本モジュールに含まれる API 関数を示します。また、表 1.2 に本モジュールで必要なメモリサイズを示します。

表 1.1　API 関数一覧

関数	関数説明
R_IRDA_SCI_Open	IrDA の初期設定を行い、本モジュールを使用できる状態にします。
R_IRDA_SCI_Close	IrDA モジュールを解放します。
R_IRDA_SCI_Send	IrDA のデータ送信を開始します。
R_IRDA_SCI_Receive	IrDA のデータ受信を開始します。
R_IRDA_SCI_Control	送受信のバッファ解放などの内部処理を行います。
R_IRDA_SCI_GetVersion	本モジュールのバージョン番号を返します。

表 1.2　必要メモリサイズ

使用メモリ	サイズ	備考
ROM	2086 バイト	
RAM	184 バイト	
最大使用ユーザスタック	80 バイト	
最大使用割り込みスタック	76 バイト	

※測定時のコンフィギュレーションオプションは、コンパイル時の設定に示すデフォルト値です。
※コンパイルオプションがデフォルトの時の値です。

必要メモリサイズは C コンパイラのバージョンやコンパイルオプションにより異なります。

1.3 IrDA FIT モジュールの概要

本モジュールでは、config.h で定義している定数を設定することで、送信バッファ、受信バッファのサイズ、通信の端子を選択することができます。

送信、受信バッファのサイズは、IRDA_SCI_CFG_CHi_TX_BUFSIZ、IRDA_SCI_CFG_CHi_RX_BUFSIZ によって設定し、使用量によって変更できます。

通信の端子は、IRDA_SCI_CFG_CHi_IRTXD_SEL、IRDA_SCI_CFG_CHi_IRRXD_SEL で選択できます。定数で選択した結果は、r_irda_sci_rxXXX.h(rxXXX はご使用になる製品名)で確認することができます。

本モジュールの R_IRDA_SCI_Open 関数を呼び出すことで、IrDA 通信をするために必要な設定を行います。ユーザプログラムでの設定は、ビットレート、High パルス幅、割り込み優先レベルが可能です。これらの設定により、モジュールストップの解除を行い、IrDA を使用するための設定が必要なレジスタ値を設定します。R_IRDA_SCI_Open 関数の応答にはハンドル情報の格納アドレスが返されます。以降、R_IRDA_SCI_Close 関数を実行するまでハンドルを、チャネルの I/O レジスタ、バッファ、およびその他の重要な情報を保持する内部の構造体として、各関数から参照します。

なお、通信完了後にビットレートなど通信設定を変更する際は、R_IRDA_SCI_Close 関数を実行した後、設定変更後の情報を引数に設定して R_IRDA_SCI_Open 関数を実行してください。

データ送信をする場合は、送信データの格納先アドレス、送信サイズを設定して R_IRDA_SCI_Send 関数を呼び出してください。送信中でなければ、TDR レジスタに 1 バイト目のデータを書き込み、残りのデータを送信バッファに保存します。送信中であれば、全ての送信データを送信バッファに保存し、TXI 割り込み処理内で順にデータを TDR レジスタに書き込みます。TXI 割り込みでは、送信バッファに保存されているデータサイズを確認しながら、送信バッファのデータを TDR レジスタに書き込みます。TEI 割り込みは、送信バッファにデータがない状態の TXI 割り込みで許可され、TEI 割り込み処理からコールバック関数を呼び出します。

データの受信は、RXI 割り込みを使用します。RXI 割り込みでは受信バッファに格納した後、RXI 割り込み処理からコールバック関数を呼び出します。受信バッファにあるデータは、データを格納する領域のアドレスと読み出しサイズを指定して R_IRDA_SCI_Receive 関数を使って読み出します。通信中に通信エラーが発生した場合は、ERI 割り込みが発生します。ERI 割り込みで通信エラーの種類と受信データを格納した後、割り込み処理からコールバック関数を呼び出します。

1.4　状態遷移図

本モジュールには、未初期化状態、アイドル状態、通信中の 3 つの状態があります。図 1.1 に本モジュールの状態遷移図を示します。

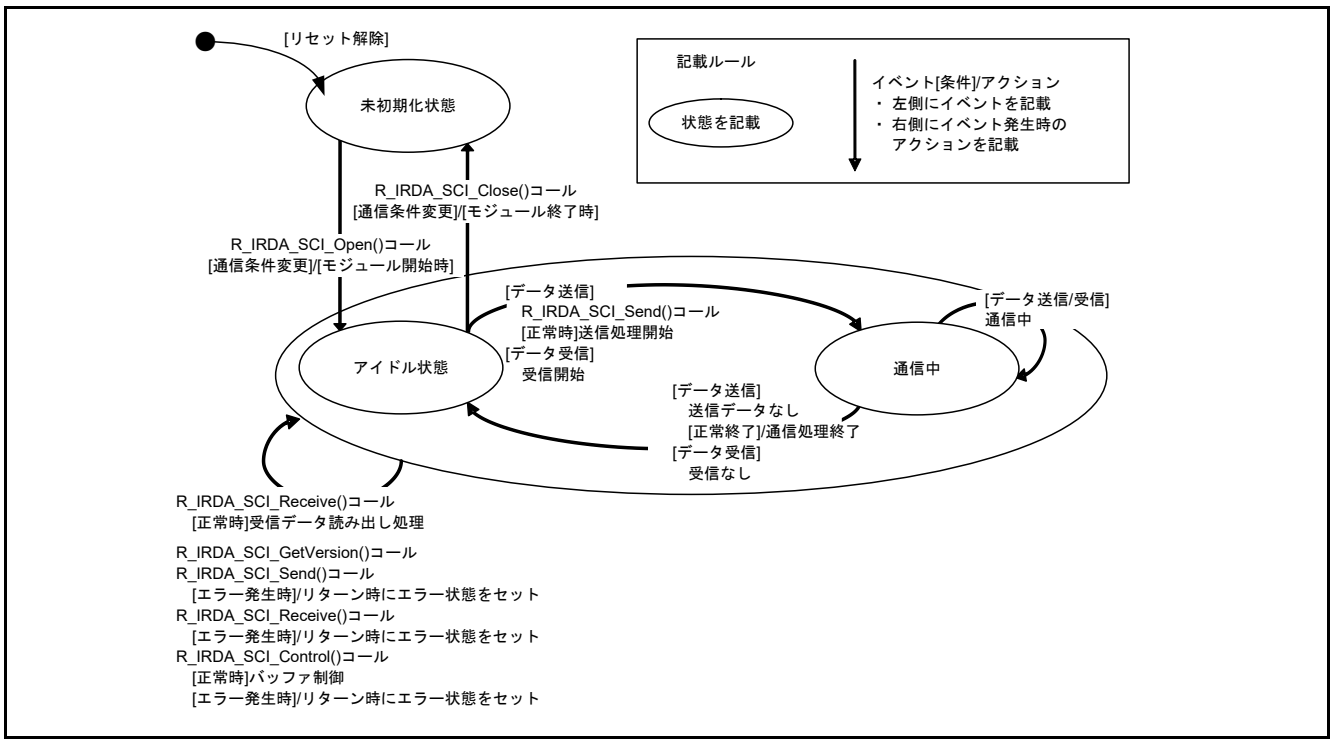

図 1.1　IrDA FIT モジュールの状態遷移図

2. API 情報

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- IrDA インタフェース

2.2 ソフトウェアの要求

本 API は以下のパッケージに依存しています。

- r_bsp
- r_byteq

2.3 サポートされているツールチェイン

本 API は下記ツールチェインで動作確認を行っています。

- Renesas RX Toolchain v.3.06

2.4 ヘッダファイル

すべての API 呼び出しとそれをサポートするインタフェース定義は r_irda_sci_rx_if.h に記載しています。コンパイル時の設定可能オプションは、r_irda_sci_rx_config.h に含まれています。ユーザアプリケーションでは両ファイルをインクルードする必要があります。

2.5 整数型

このプロジェクトは ANSI C99 を使用しています。これらの型は stdint.h で定義されています。

2.6 コンパイル時の設定

本 API のコンフィギュレーションオプションの設定は、r_irda_sci_rx_config.h で行います。オプション名および設定値に関する説明を、下表に示します。

Configuration options in r_irda_sci_rx_config.h	
#define IRDA_SCI_CFG_PARAM_CHECKING (1)	各関数の最初にあるパラメータチェック処理(引数の内容を確認する処理)をコード出力するか、しないかを選択します。 “0”の場合、パラメータチェック処理のコードを生成しません。 “1”の場合、パラメータチェック処理のコードを生成し、実行します。
#define IRDA_SCI_CFG_CHi_INCLUDED i=0～12 ※i = 5 のデフォルト値は“1” ※i = 0～4, 6～12 のデフォルト値は“0”	該当するチャネルを使用するか、しないかを選択します。 “0”の場合、該当チャネルに関する処理のコードを生成しません。 “1”の場合、該当チャネルに関する処理のコードを生成します。
IRDA_SCI_CFG_CHi_IRTXD_SEL i=0～12 ※i = 5 のデフォルト値は“1”	各チャネルの IRTXD 端子を選択します。 ch5 の値は、“1”(PC2), “2”(PA3), “3”(PA2)のいずれかが選択できます。
IRDA_SCI_CFG_CHi_IRRXD_SEL i=0～12 ※i = 5 のデフォルト値は“1”	各チャネルの IRRXD 端子を選択します。 ch5 の値は、“1”(PC3), “2”(PA4)のいずれかが選択できます。
#define IRDA_SCI_CFG_CHi_IRTXD_INACTIVE_LEVEL i=0～12 ※i = 5 のデフォルト値は“1”	R_IRDA_SCI_Close 関数を実行した際、選択した IRTXD 端子の端子レベルを示します。 “0”の場合、選択した IRTXD 端子は”Low”を出力します。 “1”の場合、選択した IRTXD 端子は”High”を出力します。
#define IRDA_SCI_CFG_CHi_IRRXD_INACTIVE_LEVEL i=0～12 ※i = 5 のデフォルト値は“1”	R_IRDA_SCI_Close 関数を実行した際、選択した IRRXD 端子の端子レベルを示します。 “0”の場合、選択した IRRXD 端子は”Low”を出力します。 “1”の場合、選択した IRRXD 端子は”High”を出力します。
#define IRDA_SCI_CFG_CHi_DATA_POLARITY i=0～12 ※i = 5 のデフォルト値は“1”	IRTXD 端子が通信時に出力する High パルス幅を選択します。 選択する値は、IrDA 制御レジスタの IrDA クロック選択ビット(IRCR.IRCKS)と同じです。ビットの説明を確認して設定してください。
#define IRDA_SCI_CFG_CHi_TX_BUFSIZ (80) i=0～12 ※デフォルト値は“80”	各チャネルの送信キューとして使用されるバッファのサイズを指定します。 対応するチャネルの IRDA_SCI_CFG_CHi_INCLUDED が“0”に設定されているときには、バッファは確保されません。
#define IRDA_SCI_CFG_CH0_RX_BUFSIZ (80) i=0～12 ※デフォルト値は“80”	各チャネルの受信キューとして使用されるバッファのサイズを指定します。 対応するチャネルの IRDA_SCI_CFG_CHi_INCLUDED が“0”に設定されているときには、バッファは確保されません。

2.7 引数

本 API の関数で使用する引数の構造体を示します。この構造体は、API 関数の外部宣言とともに r_irda_sci_rx_if.h に記載しています。

```
typedef struct st_irda_sci
{
    uint32_t    baud_rate;      /* 通信ビットレート                                    */
    uint8_t     clk_out_width;  /* IrDA IRTXD 端子の High パルスの出力幅設定値      */
    uint8_t     int_priority;   /* TXI, TEI, RXI ,ERI 割り込み優先レベル; 1=low, 15=high */
} irda_sci_t;

typedef struct st_irda_sci_ch_ctrl * irda_sci_hdl_t;    /* IrDA チャネルハンドル   */
```

構造体 irda_sci_hdl_t は、チャネルごとに自動変数、グローバル変数いずれかで宣言してください。R_IRDA_SCI_Open 関数の呼び出し時に指定した構造体変数を使用して動作します。また、この構造体は、R_IRDA_SCI_Open 関数を実行した後、R_IRDA_SCI_Close 関数が実行されるまで領域を保持してください。

2.8 戻り値

API 関数の戻り値を示します。この列挙型は、API 関数の外部宣言とともに r_irda_sci_rx_if.h で記載しています。

```
typedef enum /* IrDA API のステータスコード */
{
    IRDA_SCI_SUCCESS,                   /* 正常処理完了 */
    IRDA_SCI_ERR_LOCK_FUNC,             /* 選択したチャネルはハードウェアロック状態 */
    IRDA_SCI_ERR_BAD_CHAN,              /* チャネル番号が有効な値でない */
    IRDA_SCI_ERR_OMITTED_CHAN,          /* r_irda_sci_rx_config.h の IRDA_SCI_CHi_INCLUDED が
                                           "0" */
    IRDA_SCI_ERR_CH_NOT_CLOSED,         /* 動作中。指定したチャネルは既に使用されている。*/
    IRDA_SCI_ERR_INVALID_ARG,           /* 構造体メンバに無効な値が含まれている */
    IRDA_SCI_ERR_NULL_PTR,              /* 指定された構造体の要素が NULL */
    IRDA_SCI_ERR_QUEUE_UNAVAILABLE, /* 送信キューまたは受信キューのいずれか、もしくは
                                           両方をオープンできない */
    IRDA_SCI_ERR_INSUFFICIENT_SPACE, /* キューにデータ全体を格納するスペースがない */
    IRDA_SCI_ERR_INSUFFICIENT_DATA,  /* 指示されたサイズのデータが受信キューにない */
} irda_sci_err_t;
```

2.9 コールバック関数

本モジュールでは、TEI, RXI ,ERI 割り込みの発生タイミングでコールバック関数を呼び出します。

コールバック関数の設定は、コールバック関数として登録したい関数のアドレスを R_IRDA_SCI_Open 関数の引数に設定してください。

コールバック関数は、引数を 1 つ持っています。この引数は構造体へのポインタで、他の FIT モジュールのコールバック関数との一貫性を保つため、void ポインタ型で定義しています。構造体は以下のような構造です。

```
typedef struct st_irda_sci_cb_args
{
    irda_sci_hdl_t          hdl;
    irda_sci_cb_event_t   event;
    uint8_t                 byte;    /* 受信データ */
} irda_sci_cb_args_t;
```

引数の hdl はチャネルハンドル、event は次の enum で定義されています。byte には、受信データが格納さ

れます。

```
typedef enum e_irda_sci_cb_event
{
    IRDA_SCI_EVT_TEI,             /* TEI interrupt occurred; transmitter is idle      */
    IRDA_SCI_EVT_RX_CHAR,        /* received a character; already placed in queue  */
    IRDA_SCI_EVT_RXBUF_OVFL,     /* rx queue is full; can't save anymore data      */
    IRDA_SCI_EVT_FRAMING_ERR,    /* receiver hardware framing error            */
    IRDA_SCI_EVT_OVFL_ERR        /* receiver hardware overrun error            */
} irda_sci_cb_event_t;
```

以下にコールバック関数のサンプルを示します。

```
void MyCallback(void *p_args)
{
     irda_sci_cb_args_t *args;
     args = (irda_sci_cb_args_t *)p_args;
     if (IRDA_SCI_EVT_RX_CHAR == args->event)
     {
            // from RXI interrupt; character placed in queue is in args->byte
            nop();
     }
#if SCI_CFG_TEI_INCLUDED
     else if (IRDA_SCI_EVT_TEI == args->event)
     {
            // from TEI interrupt; transmitter is idle
            // possibly disable external transceiver here
            nop();
     }
#endif
     else if (IRDA_SCI_EVT_RXBUF_OVFL == args->event)
     {
            // from RXI interrupt; receive queue is full
            // unsaved char is in args->byte
            // will need to increase buffer size or reduce baud rate
            nop();
     }
```

```
        else if (IRDA_SCI_EVT_OVFL_ERR == args->event)
        {
                // from ERI/Group12 interrupt; receiver overflow error occurred
                // error char is in args->byte
                // error condition is cleared in ERI routine
                nop();
        }
        else if (IRDA_SCI_EVT_FRAMING_ERR == args->event)
        {
                // from ERI/Group12 interrupt; receiver framing error occurred
                // error char is in args->byte; if = 0, received BREAK condition
                // error condition is cleared in ERI routine
                nop();
        }
}
```

2.10 FIT モジュールの追加方法

本モジュールは、e2 studio で、使用するプロジェクトに追加する必要があります。

プロジェクトへの追加方法は、FIT プラグインを使用する方法と、手動で追加する方法があります。

FIT プラグインを使用すると、簡単にプロジェクトに FIT モジュールを追加でき、またインクルードファイルパスも自動的に更新できます。このため、プロジェクトへ FIT モジュールを追加する際は、FIT プラグインの使用を推奨します。

FIT プラグインを使用して FIT モジュールを追加する方法は、アプリケーションノート「e2studio に組み

込む方法(R01AN1723JU)」の「3. FIT プラグインを使用して FIT モジュールをプロジェクトに追加する方法」を参照してください。

FIT プラグインを使用せず手動で FIT モジュールを追加する方法は、「4. 手作業で FIT モジュールをプロジェクトに追加する方法」を参照してください。

FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685JU)」を参照してください。

2.11　for 文、while 文、do while 文について

本モジュールでは、レジスタの反映待ち処理等で for 文、while 文、do while 文（ループ処理）を使用しています。これらループ処理には、「WAIT_LOOP」をキーワードとしたコメントを記述しています。そのため、ループ処理にユーザがフェイルセーフの処理を組み込む場合は、「WAIT_LOOP」で該当の処理を検索できます。

以下に記述例を示します。

```
while 文の例：
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}

for 文の例：
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}

do while 文の例：
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

3. API 関数

3.1 R_IRDA_SCI_Open ()

この関数は、IrDA を動作させるための SCI の設定、割り込みの許可、IRTXD、IRRXD 端子のポート設定、他の API 関数で使用できるようにチャネルハンドルを返す関数です。

Format

```
irda_sci_err_t R_IRDA_SCI_Open(uint8_t const               chan,
                               irda_sci_t * const          p_cfg,
                               void                        (* const p_callback)(void *p_args),
                               irda_sci_hdl_t * const      p_hdl);
```

Parameters

chan	IrDA FIT モジュールで使用する SCI のチャネル (使用できるチャネルは ch5)
p_cfg	構造体（2.API 情報の“引数”を参照）のポインタ。
p_callback	受信完了または受信エラーの検出、送信終了（TEI）の条件で発生する割り込みから呼び出される関数のポインタ
p_hdl	チャネルのハンドルのポインタ

Return Values

```
IRDA_SCI_SUCCESS:                /* 正常処理完了*/
IRDA_SCI_ERR_LOCK_FUNC:          /* 選択したチャネルはハードウェアロック状態 */
IRDA_SCI_ERR_BAD_CHAN:           /* チャネル番号が有効な値でない*/
IRDA_SCI_ERR_OMITTED_CHAN:       /* r_irda_sci_rx_config.h の IRDA_SCI_CFG_CHi_
                                    INCLUDED が"0" */
IRDA_SCI_ERR_CH_NOT_CLOSED:      /* 動作中。指定したチャネルは既に使用されている */
IRDA_SCI_ERR_INVALID_ARG:        /* 構造体の要素に無効な値が含まれている*/
IRDA_SCI_ERR_QUEUE_UNAVAILABLE:  /* 送信キューまたは受信キューのいずれか、もしくは
                                    両方をオープンできない */
```

Properties

r_irda_sci_rx_if.h にプロトタイプ宣言されています。

Description

IrDA 通信を開始するため、次に示す初期設定をし、他の API 関数で使用するハンドルを*p_hdl に返します。

・IrDA 機能を有効、通信端子の極性を設定する。(IRCR レジスタの IRE ビットを"1"、IRRXINV、IRTXINV
ビットを設定)

・SCI のチャネル 5 の関連レジスタの初期設定を行う。
(調歩同期式モード、ABCS ビット="0"、2 ストップビット)

・SCI5 の送信を許可する(TE ビットを"1")

・IRRXD5 端子の Pmn 端子制御レジスタ (PmnPFS) (m = C, 4、n = 2, 3, 4)を設定した後、ポートモード

レジスタ(PMR)を周辺機能に設定(PMR レジスタの該当ビットを”1”)
・18 / (16 × SCI5 のビットレート) を待つ
・IRTXD 出力の High パルス幅を設定する。(IRCR レジスタの IRCKS ビットを設定)
・RXI 割り込み、ERI 割り込みを許可する。(IER レジスタの該当ビットを”1”)
・受信を許可する。(SCR レジスタの RE ビットを”1”)
・IRTXD5 端子の Pmn 端子制御レジスタ (PmnPFS)を設定した後、ポートモードレジスタ(PMR)を周辺機能に設定(PMR レジスタの該当ビットを”1”)
・TXI 割り込みを許可する。(IER レジスタの該当ビットを”1”)

Reentrant

この関数は、再入することはできません。

Example

```
irda_sci_hdl_t Console;

void main(void)
{
    irda_sci_err_t err;
    irda_control config;

    config.baud_rate = 115200;
    config.clk_out_width = IRDA_SCI_OUT_WIDTH_3_16;
    config.int_priority = 2; /* 1=lowest, 15=highest */
    err = R_IRDA_SCI_Open(IRDA_SCI_CH5, &config, MyCallback, &Console);
    if (IRDA_SCI_SUCCESS != err)
    {
        while(1) { };
    }

            :
            :

}
```

※コールバック関数は、コールバック関数の節から引用してください。

Special Notes:

本 API 関数は、ボードサポートパッケージ(BSP)の mcu_info.h で定義されている BSP_PCLKB_HZ に合わせて、SMR.CKS および SEMR.ABCS、BRR の最適値を計算するアルゴリズムを組み込んでいます(注 1)。

また、IRTXD 出力の High パルス幅が仕様範囲内(注 2)になっていることも確認しています。

(注 1)SCI のすべての周辺モジュールクロックとビットレートを組み合わせ、組み合わせた結果すべてで低いビットエラーの発生率を保証するものではありません。

(注 2) 最小：1.41μs。
最大：（3/16 ＋ 2.5%）× ビットレート、または（3/16 × ビットレート）+ 1.08μs。

3.2 R_IRDA_SCI_Close ()

この関数は、IrDA を停止するため、チャネルハンドルで指定したチャネルの SCI の初期設定、割り込みの禁止、IRTXD、IRRXD 端子のポート設定を実行する関数です。

Format

```
irda_sci_err_t R_IRDA_SCI_Close(
        irda_sci_hdl_t const    hdl     /* 構造体データ */
)
```

Parameters

p_hdl　　チャネルのハンドル

Return Values

IRDA_SCI_SUCCESS:　　*/* 正常処理完了*/*

IRDA_ SCI_ERR_NULL_PTR:　　*/*指定された要素が NULL */*

Properties

r_irda_sci_rx_if.h にプロトタイプ宣言されています。

Description

IrDA 通信を停止するため、次に示す初期化処理を行います。

・IRTXD5 端子の Pmn 端子制御レジスタ (PmnPFS)をポートに設定した後、ポートモードレジスタ (PMR)
　をポートに設定(PMR レジスタの該当ビットを”0”)

・TXI 割り込み、TEI 割り込みを禁止する。(IER レジスタの該当ビットを”0”)

・受信を禁止する。(SCR レジスタの RE ビットを”0”)

・RXI 割り込み、ERI 割り込みを禁止する。(IER レジスタの該当ビットを”0”)

・IRTXD 出力の High パルス幅 (IRCR レジスタの IRCKS ビット) を初期設定する。

・IRRXD5 端子の Pmn 端子制御レジスタ (PmnPFS)をポートに設定した後、ポートモードレジスタ (PMR)
　をポートに設定(PMR レジスタの該当ビットを”0”)

・SCI5 の送信を禁止する(TE ビットを"0")

・SCI のチャネル 5 の関連レジスタの初期化を行う。

・TXI,TEI,RXI,ERI の割り込み要求ビットをクリア

・IrDA 機能を無効に、通信端子の極性を初期値に設定する。(IRCR レジスタの IRE ビットを"0"、IRCR レジスタの IRRXINV、IRTXINV ビットを"0")

Reentrant

この関数は、再入することはできません。

Example

```
irda_sci_hdl_t Console;

void main(void)
{
    irda_sci_err_t err;
    irda_control config;

    config.baud_rate = 115200;
    config.clk_out_width = IRDA_SCI_OUT_WIDTH_3_16;
    config.int_priority = 2; /* 1=lowest, 15=highest */
    err = R_IRDA_SCI_Open(IRDA_SCI_CH5, &config, MyCallback, &Console);
    if (IRDA_SCI_SUCCESS != err)
    {
        while(1) { };
    }

            :
            :

    err = R_IRDA_SCI_Close(Console);
    if (IRDA_SCI_SUCCESS != err)
    {
        while(1) { };
    }

            :
            :

}
```

※コールバック関数は、コールバック関数の節から引用してください。

Special Notes:

送信動作中、受信動作中に本 API 関数を呼び出すと、実行中の送信、受信を中断します。

この関数は、送受信中でなく、かつ TEI 割り込みが実行された後に実行してください。

3.3 R_IRDA_SCI_Send ()

データの送信中でない(IDLE 状態)場合、送信バッファレジスタにデータを設定し、送信を開始します。データの送信中の場合は、送信データキューに送信するデータを格納し、その後に割り込み処理によって送信を行う関数です。

Format

```
irda_sci_err_t R_IRDA_SCI_Send(
        irda_sci_hdl_t const    hdl         /*  構造体データ  */
)
```

Parameters

p_hdl　　チャネルのハンドル

Return Values

IRDA_ SCI_SUCCESS:　　*/* 正常処理完了*/*

IRDA_ SCI_ERR_NULL_PTR:　　*/*指定された要素がNULL */*

IRDA_SCI_ERR_INSUFFICIENT_SPACE:　　*/*送信キューに指定したサイズの空きがない */*

Properties

r_irda_sci_rx_if.h にプロトタイプ宣言されています。

Description

この API 関数は、ハンドルで示した SCI チャネルからデータを送信するため、送信キューに送信データを格納します。この API 関数を呼び出した時、別のデータ送信中でなければ、送信を開始する処理を行います。

送信完了時、R_SCI_Open()で設定したコールバック関数を処理します。

Reentrant

この関数は、再入することはできません。

Example

```
#define ONETIME_SEND_SIZE 16

irda_sci_hdl_t Console;
uint8_t data_send_buf[ONETIME_SEND_SIZE] = {80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95};

void main(void)
{
    irda_sci_err_t err;
    irda_control config;
    uint16_t cnt;

    config.baud_rate = 115200;
    config.clk_out_width = IRDA_SCI_OUT_WIDTH_3_16;
    config.int_priority = 2; /* 1=lowest, 15=highest */
    err = R_IRDA_SCI_Open(IRDA_SCI_CH5, &config, MyCallback, &Console);
    if (IRDA_SCI_SUCCESS != err)
    {
        while(1) { };
    }

    /* Get the size of the send buffer, if there is free space, passing the transmitted data. */
    R_IRDA_SCI_Control(Console, IRDA_SCI_CMD_TX_Q_BYTES_FREE, (void *)&cnt);
    if (cnt - ONETIME_SEND_SIZE > 0)
    {
        /* Pass the transmitted data. If transmission idle, and starts transmission. */
        err = R_IRDA_SCI_Send(Console,&data_send_buf[0], ONETIME_SEND_SIZE);
        if (IRDA_SCI_SUCCESS != err)
        {
            while(1) { };
        }
    }
}
    ※コールバック関数は、コールバック関数の節から引用してください。
            :
            :
```

Special Notes:

送信完了時に実行するコールバック関数については、2.API 情報の”コールバック関数”を参照ください。

3.4 R_IRDA_SCI_Receive ()

この API 関数は、RXI 割り込みで受信キューに格納したデータを読み出す関数です。

Format

```
irda_sci_err_t R_IRDA_SCI_Receive(
        irda_sci_hdl_t const    hdl         /*  構造体データ  */
)
```

Parameters

p_hdl　　チャネルのハンドル

Return Values

IRDA_ SCI_SUCCESS:　　*/* *正常処理完了*/*

IRDA_ SCI_ERR_NULL_PTR:　　*/*指定された要素が NULL */*

IRDA_SCI_ERR_INSUFFICIENT_DATA:　　*/* *受信キューに指定したサイズのデータがない */*

Properties

r_irda_sci_rx_if.h にプロトタイプ宣言されています。

Description

この API 関数は、ハンドルで示した SCI チャネルの受信キューから受信データを読み出します。受信キューに指定したサイズのデータが存在しない場合、エラーで応答され、受信を待ちません。

受信中に通信エラーが発生した場合、R_SCI_Open()で設定したコールバック関数を処理します。そのため、この API 関数ではエラーが発生した時の受信データが読み出されることはありません。

Reentrant

この関数は、再入することはできません。

Example

```
irda_sci_hdl_t Console;
uint8_t data_recv_buf[80];

void main(void)
{
    irda_sci_err_t err;
    irda_control config;
    uint16_t cnt;

    config.baud_rate = 115200;
    config.clk_out_width = IRDA_SCI_OUT_WIDTH_3_16;
    config.int_priority = 2; /* 1=lowest, 15=highest */
    err = R_IRDA_SCI_Open(IRDA_SCI_CH5, &config, MyCallback, &Console);
    if (IRDA_SCI_SUCCESS != err)
    {
        while(1) { };
    }

    /* Whether the buffer is receiving data, I want to check. */
    R_IRDA_SCI_Control(Console, IRDA_SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, (void *)&cnt);
    if (0 != cnt)
    {
        /* Retrieve the data of the size stored. */
        err = R_IRDA_SCI_Receive(Console,&data_recv_buf[cnt_data],cnt);
        if (IRDA_SCI_SUCCESS != err)
        {
            while(1) { };
        }
    }
}
    ※コールバック関数は、コールバック関数の節から引用してください。
            :
            :
```

Special Notes:

受信中に通信エラーが発生した場合に実行するコールバック関数については、2.API 情報の”コールバック関数”を参照ください。

3.5 R_IRDA_SCI_Control ()

この API 関数は、送信、受信のバッファの状態を確認する関数です。コマンドによって、バッファの使用サイズを確認でき、バッファのクリアができます。

Format

```
irda_sci_err_t R_IRDA_SCI_Control(
    irda_sci_hdl_t const    hdl,        /*  構造体データ  */
    irda_sci_cmd_t const    cmd,        /*  実行するコマンド  */
    void                    *p_args     /*  引数へのポインタ(内容はコマンドによって異なる) */
)
```

Parameters

p_hdl　　チャネルのハンドル

cmd　　実行するコマンド(下記参照)

p_args　　引数へのポインタで、格納内容はコマンドによって異なる。void*型にキャストしています。

cmd に設定できるコマンドは次の通り。

```
typedef enum e_irda_sci_cmd
{
    IRDA_SCI_CMD_TX_Q_FLUSH,                /* flush transmit queue                          */
    IRDA_SCI_CMD_RX_Q_FLUSH,                /* flush receive queue                           */
    IRDA_SCI_CMD_TX_Q_BYTES_FREE,           /* get count of unused transmit queue bytes */
    IRDA_SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, /* get num bytes ready for reading               */
} irda_sci_cmd_t;
```

コマンドが IRDA_SCI_CMD_TX_Q_BYTES_FREE 、IRDA_SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ の場合、p_args は uint16_t 型のカウント値を格納する変数のポインタになります。

Return Values

IRDA_SCI_SUCCESS:　　*/* 正常処理完了*/*

IRDA_SCI_ERR_NULL_PTR:　　*/*指定された要素が NULL */*

IRDA_SCI_ERR_INVALID_ARG:　　*/* cmd または p_args は無効な値 */*

Properties

r_irda_sci_rx_if.h にプロトタイプ宣言されています。

Description

この API 関数は、ドライバの状態の読み出しに使用されます。

Reentrant

この関数は、再入することはできません。

Example

```
void MyCallback(void *p_args)
{
    irda_sci_cb_args_t *args;
    args = (irda_sci_cb_args_t *)p_args;
    if (args->event == IRDA_SCI_EVT_RX_CHAR)
    {
        // from RXI interrupt; character placed in queue is in args->byte
        nop();
    }
    else if (args->event == IRDA_SCI_EVT_TEI)
    {
        /* The received data, is an invalid value. Delete the received data. */
        R_IRDA_SCI_Control(Console, IRDA_SCI_CMD_TX_Q_FLUSH,(void *)NULL);
    }
    else if (args->event == IRDA_SCI_EVT_RXBUF_OVFL)
    {
        /* The received data, is an invalid value. Delete the received data. */
        R_IRDA_SCI_Control(Console, IRDA_SCI_CMD_RX_Q_FLUSH,(void *)NULL);
    }
    else if (args->event == IRDA_SCI_EVT_OVFL_ERR)
    {
        /* The received data, is an invalid value. Delete the received data. */
        R_IRDA_SCI_Control(Console, IRDA_SCI_CMD_RX_Q_FLUSH,(void *)NULL);
    }
    else if (args->event == IRDA_SCI_EVT_FRAMING_ERR)
    {
        /* The received data, is an invalid value. Delete the received data. */
        R_IRDA_SCI_Control(Console, IRDA_SCI_CMD_RX_Q_FLUSH,(void *)NULL);
    }
}
```

他のコマンドを使用する方法は、R_IRDA_SCI_Send 関数および R_IRDA_SCI_Receive 関数の“Example”を参照してください。

Special Notes:

なし

3.6 R_IRDA_SCI_GetVersion ()

この関数は本モジュールのバージョン番号を返します。

Format

uint32_t　R_IRDA_SCI_GetVersion(void)

Parameters

なし

Return Values

バージョン番号

Properties

r_irda_sci_rx_if.h にプロトタイプ宣言されています。

Description

この関数はこのモジュールのバージョン番号を返します。バージョン番号は符号化され、最上位の 2 バイトがメジャーバージョン番号を、最下位の 2 バイトがマイナーバージョン番号を示しています。

Reentrant

この関数は、再入可能な関数です。

Example

```
void main(void)
{
    uint32_t version;
        :
    version = R_IRDA_GetVersion();
    while(1) { };
}
```

Special Notes:

この関数は“#pragma inline”を使用してインライン化されています。

4. 提供するモジュール

提供するモジュールは、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。

5. 参考ドキュメント

ユーザーズマニュアル：ハードウェア
（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
RX ファミリコンパイラ CC-RX V2.01.00 ユーザーズマニュアル RX コーディング編（R20UT2748JJ）
（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。）

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com

お問合せ先
http://japan.renesas.com/contact/

改訂記録	RX ファミリ　IrDA モジュール Firmware Integration Technology

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2014.12.01	—	初版発行
1.01	2015.1.27	14,16,18	R_IRDA_SCI_Close 関数の Example を修正 ・2 つ目の if 文の余分な“}”を削除 R_IRDA_SCI_Send 関数の Example を修正 ・2 つ目の if 文に“}”を追加 R_IRDA_SCI_Receive 関数の Example を修正 ・2 つ目の if 文に“}”を追加
1.02	2024.11.15	6 11 プログラム	「2.3 サポートしているツールチェイン」の動作用のツールチェインを更新 「2.11 for 文、while 文、do while 文について」の章を追加。 WAIT_LOOP コメントを追加。

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。

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。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、お客様の責任において、お客様の機器・システムを設計ください。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. 当社製品または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. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 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。
6. 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。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 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。
9. 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。
10. 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。
11. 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。
12. 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。
13. 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。
14. 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.5.0-1　2020.10)

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。

© 2024 Renesas Electronics Corporation. All rights reserve