RX ファミリ

ELC モジュール Firmware Integration Technology

要旨

本アプリケーションノートは、Firmware Integration Technology (FIT)を使用した ELC モジュールについて説明します。本モジュールはイベントリンクコントローラ(ELC)を使用して、モジュール間のリンクを行います。以降、本モジュールを ELC FIT モジュールと称します。

対象デバイス

- RX113 グループ
- RX130 グループ、RX140 グループ
- RX230 グループ、RX231 グループ、RX23W グループ
- RX65N 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。

対象コンパイラ

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

各コンパイラの動作確認内容については「5.2 動作確認環境」を参照してください。

関連ドキュメント

- ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)

目次

1. 概要

ELC FIT モジュールは、各モジュール間から出力されるイベントリンク信号を他のモジュールに伝えるための設定を提供します。

1.1 ELC FIT モジュールとは

本モジュールは API として、プロジェクトに組み込んで使用します。ELC FIT モジュールの組み込み方については、「2.12 FIT モジュールの追加方法」を参照してください。

1.2 ELC FIT モジュールの概要

ELC FIT モジュールを使用する場合、以下の手順で初期設定し、ELC を動作させます。

手順 1 イベントリンク先のモジュールの初期設定をします。

手順 2 イベントリンク元のモジュールとイベントリンク先のモジュールのイベントリンクを設定します。

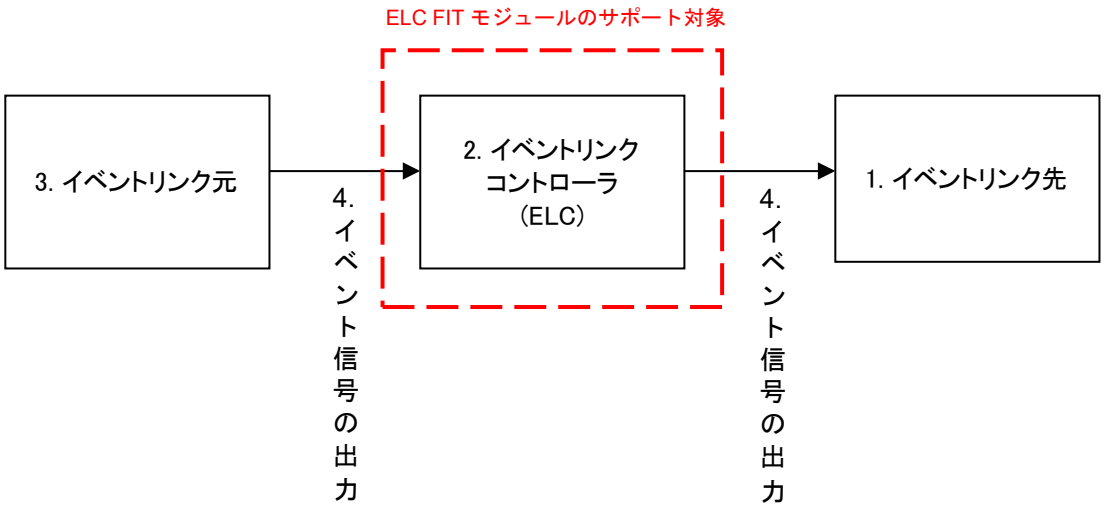
手順 3　イベントリンク元のモジュールを初期設定し、起動させます。（注 1）

手順 4 イベントリンク元のモジュールからイベント信号がイベント先モジュールに出力され、事前に設定した動作を開始します。

注1.　イベントリンク元に RTC または LVD を使用する場合は、RTC または LVD の設定を行った後、ELC の設定（手順 2）を行ってください。

ELC FIT モジュールでは、手順 2 のイベントリンク元のモジュールとイベントリンク先のモジュールのイベントリンクの設定をサポートします。手順 1、および手順 3 については、ユーザが個別に設定する必要があります。

図 1.1 に ELC の概要と設定手順を示します。



ELC FITモジュールの概要と設定手順

手順1. イベントリンク先の設定
　初めに、イベントリンク先の設定を行います。ポートに対してイベントリンクを設定する場合は、対応するポートのPODRレジスタやPDRレジスタを設定します。
手順2. イベントリンクコントローラ(ELC)の設定
　イベントリンク先、イベントリンク元間のイベントリンクの設定をします。
　本モジュールは、このイベントリンクの設定をサポートします。
手順3. イベントリンク元の設定
　イベントリンク元の設定を行い、起動させます。
手順4. イベント信号の出力
　イベントリンク元から、ELCにイベント信号が出力されます。ELCで設定されたイベントリンク先へイベント信号が伝わり、事前に設定した動作を開始します。

図 1.1　ELC FIT モジュールの概要

1.3　API の概要

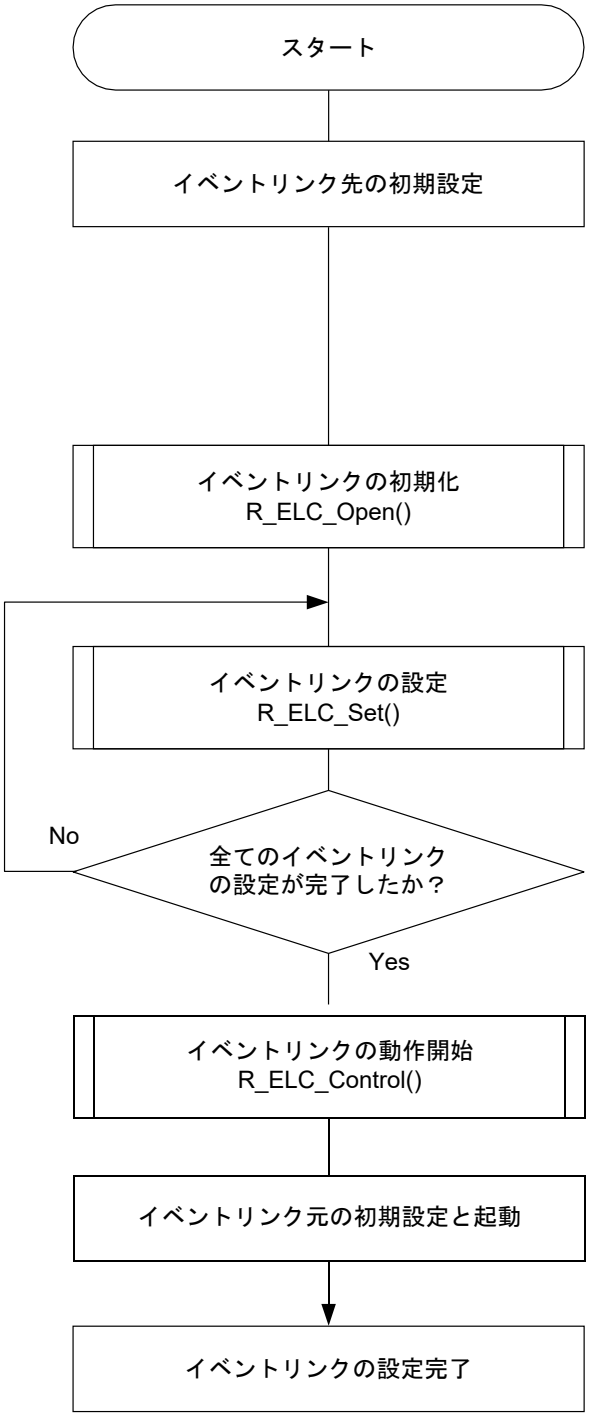
表 1.1 に本モジュールに含まれる API 関数を示します。

表 1.1　API 関数一覧

関数	関数説明
R_ELC_Open	ELC モジュールを初期化します。
R_ELC_Set	イベントリンク元のイベント信号とイベントリンク先のモジュールを接続、およびイベント発生時の動作を設定します。
R_ELC_Control	ELC モジュールの制御をおこないます。 ・イベントリンクの開始／停止 ・イベントリンクの設定解除 ・ソフトウェアイベントの発生 ・ポートバッファへの書き込み ・ポートバッファの読み出し
R_ELC_Close	ELC モジュールを停止します。
R_ELC_GetVersion	ELC FIT モジュールのバージョンを返します。

1.4　処理例

図 1.2 に ELC FIT モジュールを使用した、イベントリンクの設定例を示します。



[1] イベントリンク先の初期設定
　イベントリンク先の初期設定をします。
　ポートに対してイベントリンクを設定する場合は、対応するポートの下記のレジスタを設定します。
　PODRレジスタ：　出力に設定したポートの初期値を設定します。
　PDRレジスタ　：　ポートを入力、または出力に設定します。

[2] イベントリンクの初期化
　イベントリンクの初期化を行います。R_ELC_Open 関数をコールすることで、以降R_ELC_Set関数をコールすることが可能になります。

[3] イベントリンクの設定
　イベントリンク元とイベントリンク先の設定をします。

[4] イベントリンクの動作開始
　R_ELC_Control関数をコールすることでR_ELC_Set関数で設定したイベントリンクの動作を開始します。

[5] イベントリンク元の初期設定と起動
　イベントリンク元の初期設定をして起動させます。
　以降、イベントリンク元からイベント信号が出力されると、R_ELC_Set関数で設定した動作をします。

図 1.2　ELC FIT モジュールの設定例

1.5　状態遷移図

図 1.3 に本モジュールの状態遷移図を示します。

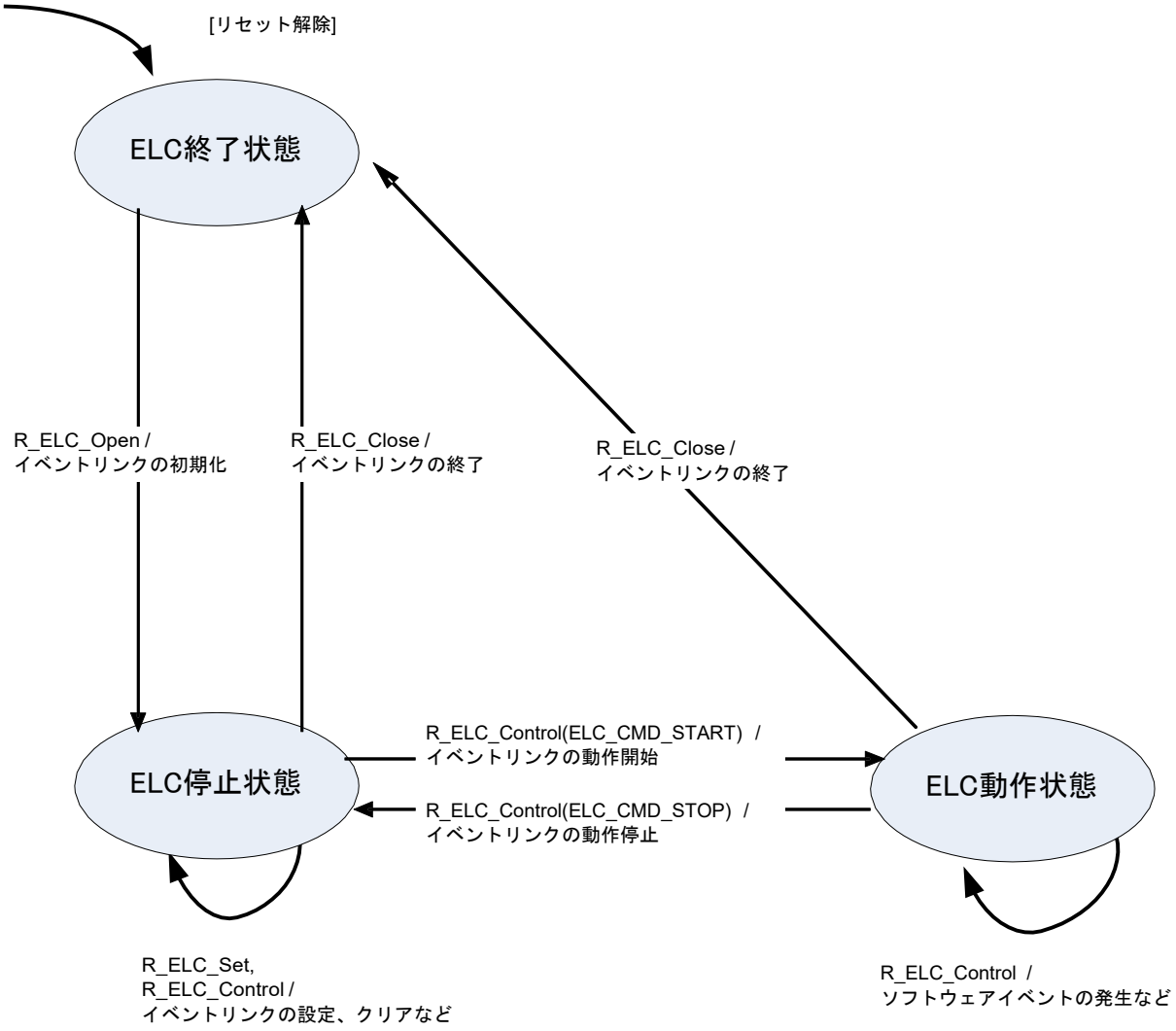


図 1.3　ELC FIT モジュールの状態遷移図

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。

- イベントリンクコントローラ(ELC)

2.2 ソフトウェアの要求

このドライバは以下のパッケージに依存しています。

- ルネサスボードサポートパッケージ（r_bsp）Rev.5.20 以上

2.3 サポートされているツールチェーン

本 FIT モジュールは「5.2 動作確認環境」に示すツールチェーンで動作確認を行っています。

2.4 使用する割り込みベクタ

R_ELC_Set 関数でイベントリンク先に ELC 割り込みを使用し、かつ割り込み優先順位に 0 以外を設定した場合、ELC 割り込みが有効になります。

表 2.1 に本 FIT モジュールが使用する割り込みベクタを示します。

表 2.1　使用する割り込みベクター覧

デバイス	割り込みベクタ
RX113、RX130、RX140	ELSR8I 割り込み（ベクタ番号:80） ELSR18I 割り込み（ベクタ番号:106）
RX230、RX231、RX23W	ELSR8I 割り込み（ベクタ番号:80） ELSR18I 割り込み（ベクタ番号:106） ELSR19I 割り込み（ベクタ番号:107）
RX65N	ELSR18I 割り込み（ベクタ番号:193）（注 1） ELSR19I 割り込み（ベクタ番号:194）（注 1）

注1. 選択型割り込み B に割り当てられている割り込みのベクタ番号については、ボードサポートパッケージ FIT モジュール(BSP モジュール)で割り当てられているデフォルト設定を記載しています。

2.5 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_elc_rx_if.h に記載しています。

2.6 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。

2.7　コンパイル時の設定

本モジュールのコンフィギュレーションオプションの設定は、r_elc_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_elc _rx_config.h	
定義	**説明**
#define ELC_CFG_PARAM_CHECKING_ENABLE ※デフォルト値は r_bsp_config.h ファイルで定義される“BSP_CFG_PARAM_CHECKING_ENABLE”の値となります。	パラメータチェック処理をコードに含めるか選択できます。 ・0：ビルド時にパラメータチェック処理をコードから省略します。 ・1：ビルド時にパラメータチェック処理をコードに含めます。 ビルド時にパラメータチェックのコードを省略することで、コードサイズを小さくすることができます。

2.8 コードサイズ

本モジュールの ROM サイズ、RAM サイズ、最大使用スタックサイズを下表に示します。

ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、ビルド時「2.7 コンパイル時の設定」のコンフィギュレーションオプションによって決まります。
下表の値は下記条件で確認しています。

モジュールリビジョン: r_elc_rx rev2.00

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00
(統合開発環境のデフォルト設定に”-lang = c99”オプションを追加)
GCC for Renesas RX 8.03.00.201904
(統合開発環境のデフォルト設定に” -std=gnu99”オプションを追加)
IAR C/C++ Compiler for Renesas RX version 4.14.01
(統合開発環境のデフォルト設定)

コンフィグレーションオプション: デフォルト設定

ROM、RAM およびスタックのコードサイズ

デバイス	分類	使用メモリ					
		Renesas Compiler		GCC		IAR Compiler	
		パラメータチェックあり	パラメータチェックなし	パラメータチェックあり	パラメータチェックなし	パラメータチェックあり	パラメータチェックなし
RX113	ROM	1295 バイト	1081 バイト	2544 バイト	2152 バイト	2277 バイト	1981 バイト
	RAM	20 バイト		20 バイト		18 バイト	
	スタック(注 1)	96 バイト		-		120 バイト	
RX130	ROM	1300 バイト	1087 バイト	2504 バイト	2120 バイト	2284 バイト	1989 バイト
	RAM	20 バイト		20 バイト		18 バイト	
	スタック(注 1)	96 バイト		-		120 バイト	
RX140	ROM	1286 バイト	1083 バイト	2512 バイト	2128 バイト	2268 バイト	1971 バイト
	RAM	20 バイト		20 バイト		18 バイト	
	スタック(注 1)	96 バイト		-		120 バイト	
RX230 RX231	ROM	1696 バイト	1482 バイト	3400 バイト	2992 バイト	3088 バイト	2736 バイト
	RAM	28 バイト		28 バイト		23 バイト	
	スタック(注 1)	96 バイト		-		120 バイト	
RX65N	ROM	1670 バイト	1480 バイト	3408 バイト	3056 バイト	3082 バイト	2777 バイト
	RAM	20 バイト		20 バイト		18 バイト	
	スタック(注 1)	108 バイト		-		128 バイト	

注1. 最大使用スタックサイズは、API 関数に割り込み処理が割り込んだ時のサイズを記載。

2.9　引数

API 関数の引数である構造体を示します。この構造体は、API 関数のプロトタイプ宣言とともにr_elc_rx_if.h に記載されています。

[イベントリンク元設定用構造体]

```
typedef struct elc_event_signal_s
{
    elc_eventlink_signal_t      event signal;                 /* イベント信号 */
    elc_port_trigger_select_t   event signal input port edge; /* 入力エッジ選択 */
    elc_single_port_select_t    event signal single port;     /* シングルポートの選択 */
    uint_8                      event signal port group bit;  /* ポートグループ指定の選択*/
} elc_event_signal_t;
```

[イベントリンク先設定用構造体]

```
typedef struct elc_link_module_s
{
    elc_ module_t                   link module;                  /* リンクする周辺モジュール */
    elc_timer_operation_select_t    link_module_timer_operation;  /* タイマ動作の選択 */
    elc_port_level_select_t         link_module_output_port_level; /*  出力ポートのレベル選択  */
    elc_single_port_select_t        link_module_single_port;      /* シングルポートの選択 */
    uint8_t                         link_module_port_group_bit;   /* ポートグループに指定する端子の選択 */
    elc_port_buffer_select_t        link_module_port_buffer;      /* ポートバッファの上書き選択 */
    uint8_t                         link_module_interrupt_level;  /*  ELC 割り込みの優先レベル */
    elc_interrupt_set _t            link_module_callbackfunc;     /*  ELC 割り込みのコールバック関数 */;
}elc_link_module_t;
```

[ポートバッファアクセス用構造体]

```
typedef struct elc_pdbf_access_s
{
    elc_portbuffer_t                select group;  /* ポートバッファグループ選択 */
    uint8_t                         value;         /*   ポートバッファへの書き込み値、または読み出し値*/;
}elc_pdbf_access_t;
```

2.10 戻り値

API 関数の戻り値を示します。この列挙型は、API 関数のプロトタイプ宣言とともに r_elc_rx_if.h で記載されています。

[エラー構造体]

```
typedef enum
{
   ELC_SUCCESS,          /* 正常終了した場合 */
   ELC_ERR_LOCK_FUNC,    /*  ELC をオープン済み */
   ELC_ERR_INVALID_ARG   /* 不正な引数が指定されていた場合 */
} elc_err_t;
```

2.11 コールバック関数

本モジュールでは、ELC 割り込み要求が発生したとき、ユーザが設定したコールバック関数を呼び出します。

コールバック関数は、「2.9 引数」に記載された構造体メンバ“link_module_callbackfunc” に、コールバック関数のアドレスを格納することで設定されます。コールバック関数が呼び出されるとき、表 2.2 に示す定数が格納された変数が、引数として渡されます。

引数の型は void ポインタ型で渡されるため、コールバック関数の引数は下記の例を参考に void 型のポインタ変数としてください。

コールバック関数内部で値を使うときはキャストして値を使用してください。

表 2.2　コールバック関数の引数一覧(enum elc_icu_t)

定数定義	説明
ELC_EVT_ICU1	ELC 割り込み 1 の割り込み処理から呼ばれたコールバック関数
ELC_EVT_ICU2	ELC 割り込み 2 の割り込み処理から呼ばれたコールバック関数(注 1)
ELC_EVT_ICU_LPT	LPT 専用 ELC 割り込みの割り込み処理から呼ばれたコールバック関数(注 2)

注1. RX113 グループ、RX130 グループ、RX140 グループでは使用できません。
注2. RX65N グループでは使用できません。

コールバック関数例：

```
void my_elc_callback(void * pdata)
{
    elc_icu_t elc_icu_number;
    elc_icu_number = *(( elc_icu_t *)pdata); //cast pointer to elc_icu_t
    …
}
```

2.12 FIT モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、スマート・コンフィグレータを使用した(1)、(3)、(5)の追加方法を推奨しています。ただし、スマート・コンフィグレータは、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(2)、(4)の方法を使用してください。

(1) e^2 studio 上でスマート・コンフィグレータを使用して FIT モジュールを追加する場合
e^2 studio のスマート・コンフィグレータを使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX スマート・コンフィグレータ ユーザーガイド: e^2 studio 編 (R20AN0451)」を参照してください。

(2) e^2 studio 上で FIT コンフィグレータを使用して FIT モジュールを追加する場合
e^2 studio の FIT コンフィグレータを使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。

(3) CS+上でスマート・コンフィグレータを使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版スマート・コンフィグレータを使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX スマート・コンフィグレータ ユーザーガイド: CS+編 (R20AN0470)」を参照してください。

(4) CS+上で FIT モジュールを追加する場合
CS+上で、手動でユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

(5) IAREW 上でスマート・コンフィグレータを使用して FIT モジュールを追加する場合
スタンドアロン版スマート・コンフィグレータを使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX スマート・コンフィグレータ ユーザーガイド: IAREW 編 (R20AN0535)」を参照してください。

3. API 関数

3.1 R_ELC_Open ()

この関数は ELC FIT モジュールを初期化し、ELC 終了状態から ELC 停止状態にする関数です。この関数は他の API 関数を使用する前に実行される必要があります。

Format

```
elc_err_t R_ELC_Open(void)
```

Parameters

なし

Return Values

ELC_SUCCESS	*/* 正常終了した場合 */*
ELC_ERR_LOCK_FUNC	*/* ELC をオープン済み */*

Properties

r_elc_rx_if.h にプロトタイプ宣言されています。

Description

イベントリンクを初期化します。また、ELC 割り込みを使用する場合は、その優先順位の設定も行います。

Example

```
volatile elc_err_t ret;

ret = R_ELC_Open();
if( ELC_SUCCESS != ret)
{
    /* 初期化に失敗した場合、エラー処理をする */
}
```

Special Notes:

・本関数をコールすると、R_ELC_Set 関数、R_ELC_Control 関数で設定した内容は
すべてクリアされます。

3.2 R_ELC_Set ()

ELC 停止状態のときに、イベントリンク元とイベントリンク先の設定をする関数です。

Format

```
elc_err_t R_ELC_Set (
  elc_event_signal_t  * const p_elc_event_signal /* リンク元設定用構造体のポインタ*/
  elc_link_module_t   * const p_elc_module       /* リンク先設定用構造体のポインタ */
)
```

Parameters

*p_elc_event_signal

イベントリンク元設定用構造体のポインタ。

イベントリンク元設定用構造体に設定する内容を表 3.1 に示します。

表 3.1　イベントリンク元設定用構造体(*p_elc_event_signal)に設定する内容

定数定義	説明
event_signal	イベントリンク元のイベント信号を設定します。 イベント信号の定義は表 5.1～表 5.2 を参照してください。
event_signal_input_port_edge	シングルポート、および入力ポートグループの有効エッジを指定します。 有効エッジの定義は表 5.7 を参照してください。 イベント信号に、シングルポートまたは入力ポートグループを選択した場合に有効です。
event_signal_single_port	シングルポートに割り当てる端子を指定します。 シングルポートの定義は表 5.5 を参照してください。 イベント信号に、シングルポートを選択した場合に有効です。
event_signal_port_group_bit	ポートグループに割り当てる端子を 8 ビットで指定します。 “1”を指定した端子がポートグループとして割り当てられます。 イベント信号に、入力ポートグループを選択した場合に有効です。

*p_elc_module

イベントリンク先設定用構造体のポインタ。

イベントリンク先設定用構造体に設定する内容を表 3.2 に示します。

表 3.2　イベントリンク先設定用構造体(*p_elc_module)に設定する内容

定数定義	説明
link_module	リンクする周辺モジュールを指定します。 指定する周辺モジュールの定義は表 5.4 を参照してください。
link_module_timer_operation	イベント信号が入力されたときのタイマ動作を指定します。 タイマ動作の定義は表 5.8 を参照ください。 リンクする周辺モジュールに MTU、TMR、または CMT を選択した場合に有効です。
link_module_output_port_level	イベント信号が入力されたときのポート出力動作を指定します。 ポート出力動作の定義は表 5.6 を参照してください。 リンクする周辺モジュールにシングルポート、または出力ポートグループを選択した場合に有効です。
link_module_single_port	シングルポートに割り当てる端子を指定します。 シングルポートの定義は表 5.5 を参照してください。 リンクする周辺モジュールにシングルポートを選択した場合に有効です。
link_module_port_group_bit	ポートグループに割り当てる端子を 8 ビットで指定します。“1”を指定した端子がポートグループとして割り当てられます。 リンクする周辺モジュールに入力ポートグループ、または出力ポートグループを選択した場合に有効です。
link_module_port_buffer	ポートバッファの上書き許可／禁止設定を指定します。 上書き許可／禁止設定の定義は表 5.9 を参照してください。 リンクする周辺モジュールに入力ポートグループを選択した場合に有効です。
link_module_interrupt_level	割り込み使用時の割り込み優先レベルを指定します。 リンクする周辺モジュールに割り込みを選択した場合に有効です。
link_module_callbackfunc	割り込み発生時にコールする、コールバック関数を指定します。 リンクする周辺モジュールに割り込みを選択した場合に有効です。

Return Values

ELC_SUCCESS　　*/* 正常終了した場合 */*

ELC_ERR _INVALID_ARG　　*/* 不正な引数が指定されていた場合 */*

Properties

r_elc_rx_if.h にプロトタイプ宣言されています。

Description

イベントリンクの設定をします。引数でイベントリンク元とイベントリンク先を指定します。

Example

設定例 1　イベントリンク元：MTU、イベントリンク先：DA

イベントリンク元を MTU、イベントリンク先を DA にする場合の設定例を以下に示します

【イベントリンク元の設定】

・**event_signal**
　イベントリンク元のイベント信号を指定します。設定例 1 では、「**MTU1・コンペアマッチ 1A**」イベント信号を指定しています。

【イベントリンク先の設定】

・**link_module**
　イベントリンク先を指定します。設定例 1 では、「**DA0**」を指定しています。

以下に、設定例 1 のソースコード例を示します。

```
volatile elc_err_t      ret;
elc_event_signal_t       event_signal_info;
elc_link_module_t    event_module_info;

ret = R_ELC_Open();                                 /* イベントリンクを初期化する */
if( ELC_SUCCESS != ret)
{
    while(1)
    {
      /* 初期化に失敗した場合、エラー処理をする */
    }
}

/* リンク元の設定 */
event_signal_info.event_signal   =   ELC_MTU1_CMP1A;   /*   リンク元のイベント信号にMTU1・コンペアマッチ 1A を指定  */

/* リンク先の設定 */
event_module_info.link_module = ELC_DA0;                    /* リンク先に DA0 を指定 */

ret = R_ELC_Set(&event_signal_info, &event_module_info);   /* リンク元のイベント信号とリンク先を
                                                              イベントリンクする */
if( ELC_SUCCESS != ret)
{
    while(1)
    {
      /* イベントリンクの設定に失敗した場合、エラー処理をする */
    }
}
```

設定例 2　イベントリンク元：シングルポート、イベントリンク先：ポートグループ

イベントリンク元をシングルポート、イベントリンク先をポートグループにする場合の設定例を以下に示します

【イベントリンク元の設定】

・**event_signal**
　イベントリンク元のイベント信号を指定します。設定例 2 では、「シングル入力ポート 2・入力エッジ検出」イベント信号を指定しています。

・**event_signal_input_port_edge**
　入力エッジ検出を指定します。設定例 2 では「立ち下がりエッジ」を指定しています。

・**event_signal_single_port**
　どのポートをシングルポートとして使用するかを指定します。設定例 2 では「PE3」を指定しています。

【イベントリンク先の設定】

・**link_module**
　イベントリンク先を指定します。設定例 2 では、「出力ポートグループ 1」(ポート B)を指定しています。

・**link_module_output_port_level**
　ポート出力時の動作を指定します。設定例 2 では「指定したポートからトグル出力」を指定しています。

・**link_module_port_group_bit**
　ポートグループとして指定したポートの、どの端子を使用するかを指定します。設定例 2 では「PB0～PB3」を指定しています。

・**link_module_port_buffer**
　PDBF レジスタへの上書きの有効/無効を指定します。設定例 2 では、上書き有効を指定しています。

以下、設定例 2 のソースコード例です。

```
volatile elc_err_t       ret;
elc_event_signal_t       event_signal_info;
elc_link_module_t        event_module_info;

ret = R_ELC_Open();                                    /* イベントリンクを初期化する */
if( ELC_SUCCESS != ret)
{
    while(1)
    {
      /* 初期化に失敗した場合、エラー処理をする */
    }
}

/* リンク元の設定 */
event_signal_info.event_signal = ELC_PORT_PSP2;     /*  リンク元のイベント信号にシングル入力ポート 2・
                                                        入力エッジ検出イベント信号を指定 */
event_signal_info.event_signal_input_port_edge = ELC_EDGE_FALLING; /* 立ち下がりエッジを指定 */
event_signal_info.event_signal_single_port =  ELC_PSB_PE3;   /*  PE3 を指定 */

  /* リンク先の設定 */
event_module_info.link_module = ELC_OUT_PGR1;        /*  リンク先に出力ポートグループ 1(ポート B)を指定 */
event_module_info.link_module_output_port_level = ELC_PORT_TOGGLE;  /* トグル出力を指定 */
event_module_info.link_module_port_group_bit =  0x0F;/*  ポートグループに PB0～PB3 を指定 */
ret = R_ELC_Set(&event_signal_info, &event_module_info);   /* リンク元のイベント信号とリンク先を
                                                              イベントリンクする */
if( ELC_SUCCESS != ret)
{
    while(1)
    {
      /* イベントリンクの設定に失敗した場合、エラー処理をする */
    }
}
```

設定例 3　イベントリンク元：ポートグループ、イベントリンク先：MTU

イベントリンク元をポートグループ 1、イベントリンク先を MTU にする場合の設定例を以下に示します

【イベントリンク元の設定】

・**event_signal**
　イベントリンク元のイベント信号を指定します。設定例 3 では「入力ポートグループ 1 (ポート B)・入力エッジ検出」イベント信号を指定しています。

・**event_signal_input_port_edge**
　入力エッジ検出を指定します。設定例 3 では「立ち下がりエッジ」を指定します。

・**link_module_port_group_bit**
　ポートグループとして指定したポートの、どの端子を使用するかを指定します。設定例 3 では「PB4～PB7」を指定しています。

【イベントリンク先の設定】

・**link_module**
　イベントリンク先を指定します。設定例 3 では「MTU1」を指定しています。

・**link_module_timer_operation**
　イベントリンク先のタイマ動作を指定します。設定例 3 では「インプットキャプチャ」を指定しています。

以下、設定例 3 のソースコード例です。

```
volatile elc_err_t      ret;
elc_event_signal_t       event_signal_info;
elc_link_module_t       event_module_info;

ret = R_ELC_Open();                                                 /* イベントリンクを初期化する */
if( ELC_SUCCESS != ret)
{
    while(1)
    {
      /* 初期化に失敗した場合、エラー処理をする */
    }
}

/* リンク元の設定 *
event_signal_info.event_signal = ELC_PORT_PGR1;     /*  リンク元のイベント信号に入力ポートグループ 1(ポート B)・
                                                    入力エッジ検出イベント信号を指定 */
event_signal_info.event_signal_input_port_edge = ELC_EDGE_FALLING; /* 立ち下がりエッジを指定 */
event_signal_info.event_signal_port_group_bit =  0xF0;/* ポートグループに PB4～PB7 を指定 */

  /* リンク先の設定 */
event_module_info.link_module = ELC_MTU1;           /*  リンク先に MTU1 を指定 */
event_module_info.link_module_timer_operation = ELC_TIMER_INPUT_CAPTURE;/* インプットキャプチャを指定 */

ret = R_ELC_Set(&event_signal_info, &event_module_info);/* リンク元のイベント信号にとリンク先を
                                                          イベントリンクする */
if( ELC_SUCCESS != ret)
{
    while(1)
    {
      /* イベントリンクの設定に失敗した場合、エラー処理をする */
    }
}
```

設定例 4　イベントリンク元：シングルポート、イベントリンク先：ELC 割り込み

イベントリンク元をシングルポート、イベントリンク先を ELC 割り込みにする場合の設定例を以下に示します

【イベントリンク元の設定】

・**event_signal**
　イベントリンク元のイベント信号を指定します。設定例 4 では、「シングル入力ポート 1・入力エッジ検出」イベント信号を指定しています。

・**event_signal_input_port_edge**
　入力エッジ検出を指定します。設定例 4 では「立ち下がりエッジ」を指定しています。

・**event_signal_single_port**
　どのポートをシングルポートとして使用するかを指定します。設定例 4 では「ポート B3」を指定しています。

【イベントリンク先の設定】

・**link_module**
　イベントリンク先を指定します。設定例 4 では「割り込み 1」を指定しています。

・**link_module_callbackfunc**
　割り込み発生時にコールされるコールバック関数を登録します。

以下、設定例 4 のソースコード例です。

```
volatile elc_err_t      ret;
elc_event_signal_t       event_signal_info;
elc_link_module_t       event_module_info;

ret = R_ELC_Open();                                                 /* イベントリンクを初期化する */
if( ELC_SUCCESS != ret)
{
    while(1)
    {
      /* 初期化に失敗した場合、エラー処理をする */
    }
}

/* リンク元の設定 */
event_signal_info.event_signal = ELC_PORT_PSP1;         /*  リンク元のイベント信号にシングル入力ポート 1・
                                                            入力エッジ検出イベント信号を指定 */

event_signal_info.event_signal_input_port_edge = ELC_EDGE_FALLING;/* 立ち下がりエッジを指定 */
event_signal_info.event_signal_single_port = ELC_PSB_PE3;          /* ポート E3 を指定 */

/* リンク先の設定 */
event_module_info.link_module = ELC_ ICU1;                         /* リンク先に ELC 割り込み 1 を指定 */
event_module_info.link_module_interrupt_level = 3;                 /* 割り込み優先レベルを 3 に設定します */
event module info.link module callbackfunc = &elc icu1 callbackfunc;    /* コールバック関数の登録 */

ret = R ELC Set(&event signal info, &event module info);           /* リンク元のイベント信号にとリンク先を
                                                                      イベントリンクする */
if( ELC_SUCCESS != ret)
{
    while(1)
    {
      /* イベントリンクの設定に失敗した場合、エラー処理をする */
    }
}

void elc_icu1_callbackfunc(void *pdata)
{
    /*  ELC 割り込み発生時のユーザ処理 */
}
```

Special Notes:

- 本関数は ELC 停止状態のときにコールしてください。
- 使用できるイベントリンク信号、リンク先の周辺モジュールは、使用するデバイスにより異なります。
- イベントリンクの動作を開始するためには、後述する R_ELC_Control 関数(ELC_CMD_START)にて ELC 動作状態にしてください。
- ELC FIT モジュールの状態については、1.5 状態遷移図を参照してください。
- リンク先の周辺モジュールに出力ポートグループを選択し、ポートグループ動作にビットローテート出力を選択する場合は、事前にポートバッファに初期値を書き込んでください。
  設定手順は、「4.4 ケースC の設定例」を参照してください。

3.3　R_ELC_Control ()

ELC 動作状態の遷移、イベントリンクの設定のクリア、ポートバッファへのアクセス、ELC のソフトウェアイベントを発生させる関数です。

Format

```
elc_err_t R_ELC_Control (
  const elc_eventlink_cmd_t command      /* コマンドの指定 */
  void                      *pdata       /* 指定したコマンドに対応した値 */
)
```

Parameters

`elc_eventlink_cmd_t　　command`

コマンドの指定。

表 3.3 に指定できるコマンド一覧表を示します。

表 3.3　コマンド一覧表

コマンドの定義	コマンドの内容
ELC_CMD_START	ELC 動作状態に遷移します。
ELC_CMD_STOP	ELC 停止状態に遷移します。
ELC_CMD_CLEAR_EVENTLINK	指定したイベントリンク設定を解除します。
ELC_CMD_WRITE_PORTBUFFER	ポートバッファに値を書き込みます。
ELC_CMD_READ_PORTBUFFER	ポートバッファの値を読み出します。
ELC_CMD_SOFTWARE_EVENT	ソフトウェアイベント信号を発生させます。

`void *pdata`

コマンドごとの引数へのポインタ。

引数に設定した void 型ポインタは、各コマンドに応じて適切な型に変換して処理されます。

表 3.4 に各コマンドに応じたポインタの設定表を示します。

表 3.4　各コマンドに応じたポインタの設定表

コマンドの定義	*pdata に設定する型	*pdata に設定する値
ELC_CMD_START	-	使用しません。 FIT_NO_PTR を設定してください。
ELC_CMD_STOP	-	使用しません。 FIT_NO_PTR を設定してください。
ELC_CMD_CLEAR_EVENTLINK	elc_link_module_t*	解除するイベントリンク先の周辺モジュールを設定した変数のポインタ。 指定する周辺モジュールの定義は表 5.4 を参照してください。
ELC_CMD_WRITE_PORTBUFFER	elc_pdbf_access_t*	アクセスするポートバッファと、書き込む値を設定した変数のポインタ 指定するポートバッファの定義は表 5.11 を参照してください。
ELC_CMD_READ_PORTBUFFER	elc_pdbf_access_t*	アクセスするポートバッファを設定した変数のポインタ。
ELC_CMD_SOFTWARE_EVENT	-	使用しません。 FIT_NO_PTR を設定してください。

Return Values

ELC_SUCCESS */* 正常終了した場合 */*

ELC_ERR _INVALID_ARG */* 不正な引数が指定されていた場合 */*

Properties

r_elc_rx_if.h にプロトタイプ宣言されています。

Description

コマンドで指定した動作を実行します。指定できるコマンドは以下になります。

・イベントリンクの開始

イベントリンクを動作状態に遷移します。戻り値は *ELC_SUCCESS* のみ返します。

```
R_ELC_Control(ELC_CMD_START, FIT_NO_PTR);     /* イベントリンクを動作状態に遷移する */
```

・イベントリンクの停止

イベントリンクを停止状態に遷移します。戻り値は *ELC_SUCCESS* のみ返します。

```
R_ELC_Control(ELC_CMD_STOP, FIT_NO_PTR);  /* イベントリンクを停止状態に遷移する */
```

・イベントリンクの設定のクリア

R_ELC_Set 関数で設定したイベントリンクをクリアします。

```
volatile elc_err_t   ret;
elc_link_module_t  elc_clear_module = ELC_ICU1;       /*  クリアするイベントリンク先に ICU1 を選択 */

ret  =  R_ELC_Control(ELC_CMD_CLEAR_EVENTLINK,  &elc_clear_module); /*  ICU1 へのイベントリンク設定を
                                                                      クリア */
```

・ポートバッファへの書き込み

ポートバッファに指定した値を書き込みます。

```
volatile elc_err_t   ret;
elc_pdbf_access_t  pdbf_access;

pdbf_access.select_group = ELC_PORT_GROUP1;              /* ポートグループ 1 を選択 */
pdbf_access.value = 0x0F;             /* ポートバッファへの書き込み値を設定 */
ret = R_ELC_Control(ELC_CMD_WRITE_PORTBUFFER, &pdbf_access);     /* ポートバッファへの書き込み */
```

・ポートバッファからの読み出し

ポートバッファから値を読み出します。

読み出した値は、引数で渡した elc_pdbf_access_t 構造体の value 要素に格納されます。

R_ELC_Control 関数の戻り値が ELC_SUCCESS であることを確認してから、値を使用してください。

```
volatile elc_err_t   ret;
uint8_t read_pdbf_value;
elc_pdbf_access_t  pdbf_access;

pdbf_access.select_group = ELC_PORT_GROUP1;              /* ポートグループ 1 を選択 */
ret = R_ELC_Control(ELC_CMD_READ_PORTBUFFER, &pdbf_access);  /* ポートバッファから読み出し */
if( ELC_SUCCESS == ret ){             /* ポートバッファ読み出し成功？ */
    read_pdbf_value = pdbf_access.value;              /* ポートバッファから読み出した値の取得 */
}
```

・ソフトウェアイベントの発生

ソフトウェアイベントを発生させることができます。

ソフトウェアイベントを発生させる場合は、先に R_ELC_Set 関数でリンク元にソフトウェアイベントを設定してください。戻り値は *ELC_SUCCESS* のみ返します。

```
     R ELC Control(ELC CMD  SOFTWARE EVENT, FIT NO PTR);
```

Example

```
volatile elc_err_t      ret;
elc_event_signal_t       event_signal_info;
elc_link_module_t       event_module_info;
elc_module_t          elc_clear_module
uint8_t                  pipr;


ret = R_ELC_Open();                                    /* イベントリンクを初期化する */
if( ELC_SUCCESS != ret)
{
    while(1)
    {
      /* 初期化に失敗した場合、エラー処理をする */
    }
}
event_signal_info.event_signal = ELC_ELC_SEG;       /* リンク元のイベント信号にソフトウェアイベントを指定 */
event_module_info.link_module = ELC_ ICU1;                  /*  リンク先に ELC 割り込み 1 を指定 */
event_module_info.link_module_interrupt_level = 3;          /*  割り込み優先レベルを 3 に設定します */
event_module_info.link_module_callbackfunc = &elc_icu1_callbackfunc;    /* コールバック関数の登録 */
ret = R_ELC_Set(&event_signal_info, &event_module_info);   /* リンク元のイベント信号にリンク先をイベント
                                                              リンクする */
if( ELC_SUCCESS != ret)
{
    while(1)
    {
      /* イベントリンクの設定に失敗した場合、エラー処理をする */
    }
}

R ELC Control(ELC CMD START,FIT NO PTR);                    /*  ELC を動作状態に遷移する */

R ELC Control(ELC CMD  SOFTWARE EVENT, FIT NO PTR);         /* ソフトウェアイベントを発生させる */

elc_clear_module  =  ELC_ICU1; /*  クリアするイベントリンク先に ELC 割り込み 1 を選択 */
R_ELC_Control(ELC_CMD_CLEAR_EVENTLINK,  &elc_clear_module); /* ELC 割り込み 1 へのイベントリンク設定をクリア  */

R ELC Control(ELC CMD STOP, FIT NO PTR);                    /*  ELC を停止状態に遷移する */
```

Special Notes:

- コマンドにイベントリンクの開始を指定する場合、ELC 停止状態のときに本関数をコールしてください。
- コマンドにイベントリンクの停止を指定する場合、ELC 動作状態のときに本関数をコールしてください。
- コマンドにソフトウェアイベントを指定する場合、ELC 動作状態のときに本関数をコールしてください。
- ELC FIT モジュールの状態については、1.5 状態遷移図を参照してください。

3.4 R_ELC_Close ()

ELC 終了状態にします。

Format

```
elc_err_t R_ELC_Close (void)
```

Parameters

なし

Return Values

ELC_SUCCESS */* 正常終了した場合 */*

Properties

r_elc_rx_if.h にプロトタイプ宣言されています。

Description

ELC モジュールを停止させます。

Example

```
R_ELC_Close();                                  /* 設定したイベントリンクの動作を終了する */
```

Special Notes:

なし

3.5 R_ELC_GetVersion ()

API のバージョンを返す関数です。

Format

```
uint32_t R_ELC_GetVersion(void)
```

Parameters

なし

Return Values

バージョン番号

Properties

r_elc_rx_if.h にプロトタイプ宣言されています。

Description

本 API のバージョン番号を返します。

Special Notes:

なし

4. 設定手順例

4.1 設定手順

ELC の設定手順を以下に示します。

設定手順：

手順 1： イベントリンク先に使用するモジュールの設定をします。
イベントリンク先に出力ポートグループのビットローテート動作を選択する場合は、ポートバッファの設定も行います。
イベントリンク元にRTC またはLVD を使用する場合は、RTC または LVD の設定を行います。

手順 2： ELC の設定をします。

手順 3： イベントリンク元に使用するモジュールの設定をします。
（RTC または LVD を使用する場合は省略）

手順 4： イベントリンク元に使用するモジュールを動作させます。

ELC FIT モジュールを使用して、以下のケース A～ケース C を設定する手順について示します。

ケース A：イベントリンク元に RTC、LVD 以外を使用する場合

ケース B：イベントリンク元に RTC または LVD を使用する場合

ケース C：イベントリンク先に出力ポートグループのビットローテート動作を選択した場合

4.2　ケース A の設定例

ケース A はイベントリンク先のモジュール設定、ELC の設定、イベントリンク元のモジュール設定の順に設定します。

ここでは、以下の条件における ELC の設定例を示します。

- 対象デバイス　　：RX231 グループ
- イベントリンク元　：MTU1・コンペアマッチ 1A イベント信号
- イベントリンク先　：S12AD（ELC からのイベント信号によりスキャン開始）

本サンプルでは、MTU FIT モジュール Rev.1.20 および S12AD FIT モジュール Rev.2.11 を使用しています。

```
#include "r_elc_rx_if.h"
#include "r_s12ad_rx_if.h"
#include "r_mtu_rx_if.h"

void main(void);
void adc_int_callback(void *p_args);

void main()
{
    mtu_timer_chnl_settings_t  my_timer_cfg;
    mtu_err_t  mtu_result;

    elc_event_signal_t event_signal_info;
    elc_link_module_t event_module_info;
    elc_err_t elc_result;

    adc_cfg_t my_adc_cfg;
    adc_ch_cfg_t my_adc_ch_cfg;
    adc_err_t  adc_result;

    /* イベントリンク先(S12AD)の設定 */
    my_adc_cfg.conv_speed = ADC_CONVERT_SPEED_DEFAULT;
    my_adc_cfg.alignment = ADC_ALIGN_RIGHT;
    my_adc_cfg.add_cnt = ADC_ADD_OFF;
    my_adc_cfg.clearing = ADC_CLEAR_AFTER_READ_OFF;
    my_adc_cfg.trigger = ADC_TRIG_SYNC_ELCTRG0N_OR_ELCTRG1N;   /* A/D 変換トリガを ELC からの
                                                                  イベント入力に指定 */
    my_adc_cfg.priority = 3;
    adc_result = R_ADC_Open(0, ADC_MODE_SS_ONE_CH, &my_adc_cfg, &adc_int_callback);

    my_adc_ch_cfg.chan_mask = ADC_MASK_CH0;
    my_adc_ch_cfg.chan_mask_groupb = ADC_MASK_GROUPB_OFF;
    my_adc_ch_cfg.priority_groupa = ADC_GRPA_PRIORITY_OFF;
    my_adc_ch_cfg.diag_method = ADC_DIAG_OFF;
    my_adc_ch_cfg.add_mask = 0;
    my_adc_ch_cfg.signal_elc = ADC_ELC_ALL_SCANS_DONE;
    adc_result = R_ADC_Control(0, ADC_CMD_ENABLE_CHANS, &my_adc_ch_cfg);
    adc_result = R_ADC_Control(0, ADC_CMD_ENABLE_TRIG, NULL);
```

```
    /* ELC の設定 */
    elc_result = R_ELC_Open();
    event_signal_info.event_signal = ELC_MTU1_CMP1A;
    event_module_info.link_module = ELC_S12AD;
    elc_result = R_ELC_Set(&event_signal_info, &event_module_info);
    /* ELC の設定を複数する場合は、R_ELC_Control 関数を実行する前に R_ELC_Set  関数をコールしてください。*/
    elc_result = R_ELC_Control(ELC_CMD_START, FIT_NO_PTR);

    /* イベントリンク元(MTU1)の設定 */
    my_timer_cfg.clock_src.source         = MTU_CLK_SRC_INTERNAL;
    my_timer_cfg.clock_src.clock_edge     = MTU_CLK_RISING_EDGE;
    my_timer_cfg.clear_src               = MTU_CLR_TIMER_A;
    my_timer_cfg.timer_a.actions.do_action = MTU_ACTION_REPEAT;
    my_timer_cfg.timer_a.freq            = 1000; //1KHz
    my_timer_cfg.timer_b.actions.do_action = MTU_ACTION_NONE;
    my_timer_cfg.timer_c.actions.do_action = MTU_ACTION_NONE;
    my_timer_cfg.timer_d.actions.do_action = MTU_ACTION_NONE;
    mtu_result = R_MTU_Timer_Open(MTU_CHANNEL_1, &my_timer_cfg, FIT_NO_FUNC);
    /* イベントリンク元(MTU)の動作を開始 */
    mtu_result = R_MTU_Control(MTU_CHANNEL_1, MTU_CMD_START, FIT_NO_PTR);

    while(1)
    {
      /* Main loop */
    }
}

void adc_int_callback(void *p_args)
{
   /*  A/D 変換終了割り込み処理 */
}
```

4.3　ケース B の設定例

ケース B はイベントリンク元の設定を ELC の設定の前に設定します。以下にイベントリンク元に RTC(周期イベント信号)、イベントリンク先に S12AD(ELC からのトリガによりスキャン開始)を設定する場合のサンプルコードを示します。

ここでは、以下の条件における ELC の設定例を示します。

- 対象デバイス　　：RX231 グループ
- イベントリンク元　：RTC 周期（1 秒）
- イベントリンク先　：S12AD（ELC からのイベント信号によりスキャン開始）

本サンプルでは、RTC FIT モジュール Rev.2.41 および S12AD FIT モジュール Rev.2.11 を使用しています。

```
#include "r_elc_rx_if.h"
#include "r_rtc_rx_if.h"
#include "r_s12ad_rx_if.h"

void main(void);
void adc_int_callback(void *p_args);
void rtc_int_callback(void *p_args);

void main()
{
   adc_cfg_t my_adc_cfg;
   adc_ch_cfg_t my_adc_ch_cfg;
   adc_err_t  adc_result;

   elc_event_signal_t event_signal_info;
   elc_link_module_t event_module_info;
   elc_err_t elc_result;

   rtc_init_t rtc_init;
   rtc_err_t rtc_result;

   /* set the current date & time to be Aug 31, 2015 (Monday) 11:59:20pm */
   tm_t init_time =
   {
     20, //Second
     59, //Minutes
     23, //Hours
     31, //Day of month
     (8-1), //Month
     115, //Years since 1900
     1, //Day of week
     0, //
     0, //Daylight savings disabled
   };
```

```
    /* イベントリンク元(RTC)の設定 */
    rtc_init.output_freq = RTC_OUTPUT_OFF;
    rtc_init.periodic_freq = RTC_PERIODIC_1_HZ;
    rtc_init.periodic_priority = 1;
    rtc_init.set_time = true;
    rtc_init.p_callback = rtc_int_callback;
    rtc_result = R_RTC_Open(&rtc_init, &init_time);

    /* イベントリンク先(S12AD)の設定 */
    my_adc_cfg.conv_speed = ADC_CONVERT_SPEED_DEFAULT;
    my_adc_cfg.alignment = ADC_ALIGN_RIGHT;
    my_adc_cfg.add_cnt = ADC_ADD_OFF;
    my_adc_cfg.clearing = ADC_CLEAR_AFTER_READ_OFF;
    my_adc_cfg.trigger = ADC_TRIG_SYNC_ELCTRG0N_OR_ELCTRG1N;   /* A/D 変換トリガを ELC からの
                                                                  イベント入力に指定 */
    my_adc_cfg.priority = 3;
    adc_result = R_ADC_Open(0, ADC_MODE_SS_ONE_CH, &my_adc_cfg, &adc_int_callback);

    my_adc_ch_cfg.chan_mask = ADC_MASK_CH0;
    my_adc_ch_cfg.chan_mask_groupb = ADC_MASK_GROUPB_OFF;
    my_adc_ch_cfg.priority_groupa = ADC_GRPA_PRIORITY_OFF;
    my_adc_ch_cfg.diag_method = ADC_DIAG_OFF;
    my_adc_ch_cfg.add_mask = 0;
    my_adc_ch_cfg.signal_elc = ADC_ELC_ALL_SCANS_DONE;
    adc_result = R_ADC_Control(0, ADC_CMD_ENABLE_CHANS, &my_adc_ch_cfg);
    adc_result = R_ADC_Control(0, ADC_CMD_ENABLE_TRIG, NULL);

    /* ELC の設定 */
    elc_result = R_ELC_Open();
    event_signal_info.event_signal = ELC_RTC_PRD;
    event_module_info.link_module = ELC_S12AD;
    elc_result = R_ELC_Set(&event_signal_info, &event_module_info);
    /* ELC の設定を複数する場合は、R_ELC_Control 関数を実行する前に R_ELC_Set  関数をコールしてください。*/
    elc_result = R_ELC_Control(ELC_CMD_START, FIT_NO_PTR);

    while(1)
    {
        /* RTC で設定した周期ごとに A/D 変換をする */
    }
}

void adc_int_callback(void *p_args)
{
    /* A/D 変換終了割り込み処理 */
}

void rtc_int_callback(void *p_args)
{
    /* 処理なし */
}
```

4.4　ケース C の設定例

ケース C は出力ポートグループの初期値を ELC の設定の前に設定します。以下にイベントリンク元にソフトウェアイベント、イベントリンク先に出力ポートグループ 1 をビットローテート動作で設定する場合のサンプルコードを示します。

```
#include “r_elc_rx_if.h”

void main(void);

void main()
{
    elc_event_signal_t   event;
    elc_link_module_t    link;
    elc_pdbf_access_t    pdbf;
    elc_err_t        elc_err;

    PORTB.PDR.BYTE = 0x0F;       /* ポートグループ 1(PORTB)の端子を出力に設定 */
    PORTB.PODR.BYTE = 0x00;      /* ポートグループ 1(PORTB)の端子を Low に設定 */

    /* ELC の設定 */
    elc_err = R_ELC_Open();

    event.event_signal = ELC_ELC_SEG;   /* イベントリンク元をソフトウェアトリガに指定 */
    link.link_module = ELC_OUT_PGR1;        /* イベントリンク先を出力ポートグループ 1 に指定 */
    link.link_module_output_port_level = ELC_PORT_ROTATE;  /* ローテート出力 */
    link.link_module_port_group_bit = (uint8_t)0x0F;           /* PB3～PB0 でローテート */

    /* イベントリンクを設定する前に PDBF1 レジスタにローテート出力の初期値を設定 */pdbf.select_group
    = ELC_PORT_GROUP1;
    pdbf.value = 0x08;
    elc_err = R_ELC_Control( ELC_CMD_WRITE_PORTBUFFER, &pdbf );

    elc err = R ELC Set( &event, &link );                       /* イベントリンクを設定 */
    elc_err =  R_ELC_Control(  ELC_CMD_START,  FIT_NO_PTR );   /* ELC 動作状態に遷移 */

    while(1)
    {
      R_ELC_Control( ELC_CMD_SOFTWARE_EVENT, FIT_NO_PTR );
      /*
      * ソフトウェアトリガを発生させるたびに、PDBF1　レジスタに設定した値が
      * PB3～PB0 の間で MSB→LSB にローテートする
      */
    }
}
```

5. 付録

5.1 定義一覧

各関数の引数で使用する定義を以下に示します。

表 5.1 イベントリンク信号の定義一覧(1/3)

定義名	説明
ELC_MTU0_CMP0A	MTU0・コンペアマッチ 0A イベント信号
ELC_MTU0_CMP0B	MTU0・コンペアマッチ 0B イベント信号
ELC_MTU0_CMP0C	MTU0・コンペアマッチ 0C イベント信号
ELC_MTU0_CMP0D	MTU0・コンペアマッチ 0D イベント信号
ELC_MTU0_CMP0E	MTU0・コンペアマッチ 0E イベント信号
ELC_MTU0_CMP0F	MTU0・コンペアマッチ 0F イベント信号
ELC_MTU0_OVF	MTU0・オーバフローイベント信号
ELC_MTU1_CMP1A	MTU1・コンペアマッチ 1A イベント信号
ELC_MTU1_CMP1B	MTU1・コンペアマッチ 1B イベント信号
ELC_MTU1_OVF	MTU1・オーバフローイベント信号
ELC_MTU1_UDF	MTU1・アンダフローイベント信号
ELC_MTU2_CMP2A	MTU2・コンペアマッチ 2A イベント信号
ELC_MTU2_CMP2B	MTU2・コンペアマッチ 2B イベント信号
ELC_MTU2_OVF	MTU2・オーバフローイベント信号
ELC_MTU2_UDF	MTU2・アンダフローイベント信号
ELC_MTU3_CMP3A	MTU3・コンペアマッチ 3A イベント信号
ELC_MTU3_CMP3B	MTU3・コンペアマッチ 3B イベント信号
ELC_MTU3_CMP3C	MTU3・コンペアマッチ 3C イベント信号
ELC_MTU3_CMP3D	MTU3・コンペアマッチ 3D イベント信号
ELC_MTU3_OVF	MTU3・オーバフローイベント信号
ELC_MTU4_CMP4A	MTU4・コンペアマッチ 4A イベント信号
ELC_MTU4_CMP4B	MTU4・コンペアマッチ 4B イベント信号
ELC_MTU4_CMP4C	MTU4・コンペアマッチ 4C イベント信号
ELC_MTU4_CMP4D	MTU4・コンペアマッチ 4D イベント信号
ELC_MTU4_OVF	MTU4・オーバフローイベント信号
ELC_MTU4_UDF	MTU4・アンダフローイベント信号
ELC_CMT_CMP1	CMT1・コンペアマッチ 1 イベント信号
ELC_TMR0_CMPA0	TMR0・コンペアマッチ A0 イベント信号
ELC_TMR0_CMPB0	TMR0・コンペアマッチ B0 イベント信号
ELC_TMR0_OVF	TMR0・オーバフローイベント信号
ELC_TMR1_CMPA1	TMR1・コンペアマッチ A1 イベント信号
ELC_TMR1_CMPB1	TMR1・コンペアマッチ B1 イベント信号
ELC_TMR1_OVF	TMR1・オーバフローイベント信号
ELC_TMR2_CMPA2	TMR2・コンペアマッチ A2 イベント信号
ELC_TMR2_CMPB2	TMR2・コンペアマッチ B2 イベント信号
ELC_TMR2_OVF	TMR2・オーバフローイベント信号
ELC_TMR3_CMPA3	TMR3・コンペアマッチ A3 イベント信号
ELC_TMR3_CMPB3	TMR3・コンペアマッチ B3 イベント信号
ELC_TMR3_OVF	TMR3・オーバフローイベント信号
ELC_RTC_PRD	RTC・周期イベント信号(注 1)
ELC_IWDT_UDF	IWDT・アンダフロー・リフレッシュエラーイベント信号
ELC_LPT_CMP0	LPT・コンペアマッチ 0
ELC_LPT_CMP1	LPT・コンペアマッチ 1

注 1. 本イベント信号を設定する場合は、設定手順が他のイベント信号と異なります。詳細は 4 設定手順の章のケース B をご参照ください。

表 5.2　イベントリンク信号の定義一覧(2/3)

使用する定義名	内容
ELC_S12AD_WMELC	S12AD・比較条件成立
ELC_S12AD_WUMELC	S12AD・比較条件不成立
ELC_SCI5_ER5	SCI5・エラー（受信エラー・エラーシグナル検出）イベント信号
ELC_SCI5_RX5	SCI5・受信データフルイベント信号
ELC_SCI5_TX5	SCI5・送信データエンプティイベント信号
ELC_SCI5_TE5	SCI5・送信完了イベント信号
ELC_RIIC0_ER0	RIIC0・通信エラー、イベント発生信号
ELC_RIIC0_RX0	RIIC0・受信データフルイベント信号
ELC_RIIC0_TX0	RIIC0・送信データエンプティイベント信号
ELC_RIIC0_TE0	RIIC0・送信終了イベント信号
ELC_RSPI0_ER0	RSPI0・エラー(モードフォルト・オーバラン・アンダラン・パリティエラー)イベント信号
ELC_RSPI0_IDLE	RSPI0・アイドルイベント信号
ELC_RSPI0_RX0	RSPI0・受信データフルイベント信号
ELC_RSPI0_TX0	RSPI0・送信データエンプティイベント信号
ELC_RSPI0_TE0	RSPI0・送信完了イベント信号
ELC_S12AD_S12AD0	S12AD・A/D 変換終了イベント信号
ELC_CMPB_CMPB0	コンパレータ B0・比較結果変化
ELC_CMPB_CMPB0_CMPB1	コンパレータ B0・B1 共通比較結果変化
ELC_LVD1_LVD1	LVD1・電圧検出イベント信号(注 1)
ELC_LVD2_LVD2	LVD2・電圧検出イベント信号(注 1)
ELC_DMAC0_DMAC0	DMAC0・転送終了イベント信号
ELC_DMAC1_DMAC1	DMAC1・転送終了イベント信号
ELC_DMAC2_DMAC2	DMAC2・転送終了イベント信号
ELC_DMAC3_DMAC3	DMAC3・転送終了イベント信号
ELC_DTC_DTC	DTC・転送終了イベント信号
ELC_CGC_OSTD	クロック発生回路・発振停止検出イベント信号
ELC_PORT_PGR1	入力ポートグループ 1・入力エッジ検出イベント信号
ELC_PORT_PGR2	入力ポートグループ 2・入力エッジ検出イベント信号
ELC_PORT_PSP0	シングル入力ポート 0・入力エッジ検出イベント信号
ELC_PORT_PSP1	シングル入力ポート 1・入力エッジ検出イベント信号
ELC_PORT_PSP2	シングル入力ポート 2・入力エッジ検出イベント信号
ELC_PORT_PSP3	シングル入力ポート 3・入力エッジ検出イベント信号
ELC_ELC_SEG	ソフトウェアイベント
ELC_DOC_DOPCF	DOC・データ演算条件成立信号

注 1. 本イベント信号を設定する場合は、設定手順が他のイベント信号と異なります。詳細は 4 設定手順の章のケース B を参照ください。

表 5.3 イベントリンク信号の定義一覧(3/3)

使用する定義名	内容
ELC_S12AD_S12AD1	S12AD1・A/D 変換終了イベント信号
ELC_CMT_CMPW	CMTW・チャネル 0・コンペアマッチ信号
ELC_TPU0_CMPA	TPU0・コンペアマッチ A イベント信号
ELC_TPU0_CMPB	TPU0・コンペアマッチ B イベント信号
ELC_TPU0_CMPC	TPU0・コンペアマッチ C イベント信号
ELC_TPU0_CMPD	TPU0・コンペアマッチ D イベント信号
ELC_TPU0_OVF	TPU0・オーバフローイベント信号
ELC_TPU1_CMPA	TPU1・コンペアマッチ A イベント信号
ELC_TPU1_CMPB	TPU1・コンペアマッチ B イベント信号
ELC_TPU1_OVF	TPU1・オーバフローイベント信号
ELC_TPU1_UDF	TPU1・アンダフローイベント信号
ELC_TPU2_CMPA	TPU2・コンペアマッチ A イベント信号
ELC_TPU2_CMPB	TPU2・コンペアマッチ B イベント信号
ELC_TPU2_OVF	TPU2・オーバフローイベント信号
ELC_TPU2_UDF	TPU2・アンダフローイベント信号
ELC_TPU3_CMPA	TPU3・コンペアマッチ A イベント信号
ELC_TPU3_CMPB	TPU3・コンペアマッチ B イベント信号
ELC_TPU3_CMPC	TPU3・コンペアマッチ C イベント信号
ELC_TPU3_CMPD	TPU3・コンペアマッチ D イベント信号
ELC_TPU3_OVF	TPU3・オーバフローイベント信号

表 5.4　イベントリンク先の周辺モジュールの定義一覧

使用する定義名	内容
ELC_MTU0	MTU0
ELC_MTU1	MTU1
ELC_MTU2	MTU2
ELC_MTU3	MTU3
ELC_MTU4	MTU4
ELC_CMT1	CMT1
ELC_ICU_LPT	ELC 割り込み（LPT 専用）
ELC_TMR0	TMR0
ELC_TMR1	TMR1
ELC_TMR2	TMR2
ELC_TMR3	TMR3
ELC_CTSU	CTSU
ELC_S12AD	S12AD
ELC_DA0	DA0
ELC_ICU1	ELC 割り込み 1
ELC_ICU2	ELC 割り込み 2
ELC_OUT_PGR1	出力ポートグループ 1
ELC_OUT_PGR2	出力ポートグループ 2
ELC_IN_PGR1	入力ポートグループ 1
ELC_IN_PGR2	入力ポートグループ 2
ELC_PSP0	シングルポート 0
ELC_PSP1	シングルポート 1
ELC_PSP2	シングルポート 2
ELC_PSP3	シングルポート 3
ELC_CGC_LOCO	クロック発生回路（クロックソースを LOCO へ切り替え）
ELC_POE	POE
ELC_CMTW0	CMTW0
ELC_TPU0	TPU0
ELC_TPU1	TPU1
ELC_TPU2	TPU2
ELC_TPU3	TPU3
ELC_S12AD1	S12AD1

表 5.5　イベント接続ポート選択定義

使用する定義名	内容
ELC_PSB_PB0	シングルポートにポート B0 を指定
ELC_PSB_PB1	シングルポートにポート B1 を指定
ELC_PSB_PB2	シングルポートにポート B2 を指定
ELC_PSB_PB3	シングルポートにポート B3 を指定
ELC_PSB_PB4	シングルポートにポート B4 を指定
ELC_PSB_PB5	シングルポートにポート B5 を指定
ELC_PSB_PB6	シングルポートにポート B6 を指定
ELC_PSB_PB7	シングルポートにポート B7 を指定
ELC_PSB_PE0	シングルポートにポート E0 を指定
ELC_PSB_PE1	シングルポートにポート E1 を指定
ELC_PSB_PE2	シングルポートにポート E2 を指定
ELC_PSB_PE3	シングルポートにポート E3 を指定
ELC_PSB_PE4	シングルポートにポート E4 を指定
ELC_PSB_PE5	シングルポートにポート E5 を指定
ELC_PSB_PE6	シングルポートにポート E6 を指定
ELC_PSB_PE7	シングルポートにポート E7 を指定

表 5.6　イベントリンク信号によるシングルポート/ポートグループ動作選択定義一覧

使用する定義名	内容
ELC_PORT_LOW	指定したポートから Low 出力
ELC_PORT_HIGH	指定したポートから High 出力
ELC_PORT_TOGGLE	指定したポートからトグル出力
ELC_PORT_BUFFER	指定したポートからポートバッファの値を出力(注 1)
ELC_PORT_ROTATE	指定したポートからビットローテート出力(注 1) (注 2)

注1. 出力ポートグループ選択時のみ選択してください。シングルポート出力では選択しないでください。
注2. イベントリンク先の周辺モジュールに出力ポートグループを選択し、ポートグループ動作にビットローテート出力を選択する場合は、事前にポートバッファに初期値を書き込んでください。
設定手順は、「4.4 ケースC の設定例」を参照してください。

表 5.7　外部入力信号のエッジ選択定義

使用する定義名	内容
ELC_EDGE_RISING	外部入力信号の立ち上がりエッジを検出
ELC_EDGE_FALLING	外部入力信号の立ち下がりエッジを検出
ELC_EDGE_RISING_AND_FALLING	外部入力信号の立ち上がり/立ち下がりの両エッジを検出

表 5.8　イベントリンク信号によるタイマ動作選択定義一覧

使用する定義名	内容
ELC_TIMER_START	タイマスタート
ELC_TIMER_RESTART	タイマリスタート
ELC_TIMER_INPUT_CAPTURE	インプットキャプチャ
ELC_TIMER_DISABLED	イベント無効

表 5.9　ポートバッファの上書き許可/禁止設定定義一覧

使用する定義名	内容
ELC_PDBF_OVERWRITE_ENABLE	ポートバッファの上書きを許可
ELC_PDBF_OVERWRITE_DISABLE	ポートバッファの上書きを禁止

表 5.10　コントロール関数で使用するコマンド定義一覧

使用する定義名	内容
ELC_CMD_START	ELC 動作状態に遷移します。
ELC_CMD_STOP	ELC 停止状態に遷移します。
ELC_CMD_CLEAR_EVENTLINK	指定したモジュールに対するイベントリンク設定を解除します。
ELC_CMD_WRITE_PORTBUFFER	ポートバッファに値を書き込みます
ELC_CMD_READ_PORTBUFFER	ポートバッファから値を読み出します。
ELC_CMD_SOFTWARE_EVENT	ソフトウェアイベント信号を発生させます

表 5.11　ポートグループ選択定義一覧

使用する定義名	内容
ELC_PORT_GROUP1	ポートグループ 1 を選択
ELC_PORT_GROUP2	ポートグループ 2 を選択

5.2　動作確認環境

本 FIT モジュールの動作確認環境を以下に示します。

表 5.12　動作確認環境 (Rev.2.01)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio Version 2021-07 IAR Embedded Workbench for Renesas RX 4.20.01
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.03.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 8.3.0.202004 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.20.1 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev2.01
使用ボード	Target board for RX140 (型名：RTK5RX140xxxxxxxxx)

表 5.13　動作確認環境 (Rev.2.00)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V7.7.0 IAR Embedded Workbench for Renesas RX 4.14.01
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.02.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 8.03.00.201904 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.14.01 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev2.00
使用ボード	Renesas Solution Starter Kit for RX23W (型名：RTK5523Wxxxxxxxxxx) Renesas Starter Kit for RX130 (型名：RTK5005130xxxxxxxx) Renesas Starter Kit for RX231 (型名：R0K505231xxxxxx) Renesas Starter Kit+ for RX65N (型名：RTK500565Nxxxxxxxx)

表 5.14　動作確認環境 (Rev.1.21)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V7.3.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.01.00
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev1.21

表 5.15　動作確認環境 (Rev.1.20)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio V6.0.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V2.07.00
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev1.20
使用ボード	Renesas Starter Kit+ for RX65N-2MB (型名：RTK50565N2SxxxxxBE) Renesas Starter Kit for RX130-512KB (型名：RTK5051308SxxxxxBE)

5.3　トラブルシューティング

(1) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file “platform.h”」エラーが発生します。

A：FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

- CS+を使用している場合
  アプリケーションノート RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」
- e^2 studio を使用している場合
  アプリケーションノート RX ファミリ e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、本 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(R01AN1685)」を参照してください。

(2) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「This MCU is not supported by the current r_elc_rx module.」エラーが発生します。

A：追加したFIT モジュールがユーザプロジェクトのターゲットデバイスに対応していない可能性があります。追加した FIT モジュールの対象デバイスを確認してください。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2016.07.01	−	初版発行
1.10	2016.10.01	1,8,10,34,36,37	RX65N に対応
1.20	2017.07.24	−	RX65N-2MB、RX130-512KB に対応
		1 8 12 41 42	「関連ドキュメント」に以下のドキュメントを追加: Renesas e^2 studio スマート・コンフィグレータ ユーザーガイド(R20AN0451) 2.6 使用する割り込みベクタ: 追加 2.12 FIT モジュールの追加方法: 変更 5.2 動作確認環境: 追加 5.3 トラブルシューティング: 追加
1.21	2019.04.01	−	機能関連 Smart Configurator での GUI によるコンフィグオプション設定機能に対応 ■内容 GUIによるコンフィグオプション設定機能に対応するため、設定ファイルを追加。
		4	「1.3 API の概要」を移動
		7	「2.5 ヘッダファイル」「2.6 整数型」を移動
		8	「2.8 コードサイズ」を変更
		40	「5.2 動作確認環境」 Rev.1.21 に対応する表を追加
2.00	2020.06.10	−	RX23W に対応 API 関数のコメントを Doxygen スタイルに変更 以下のコンパイラに対応 ・GCC for Renesas RX ・IAR C/C++ Compiler for Renesas RX
		1	「対象コンパイラ」を追加
		1	「関連ドキュメント」に以下のドキュメントを削除 Firmware Integration Technology ユーザーズマニュアル (R01AN1833) e2studio に組み込む方法 Firmware Integration Technology (R01AN1723) CS+に組み込む方法 Firmware Integration Technology (R01AN1826) Renesas e2 studio スマート・コンフィグレータ ユーザーガイド(R20AN0451)
		7	「2.2 ソフトウェアの要求」 依存する r_bsp モジュールのリビジョンを追加
		7	「2.4 使用する割り込みベクタ」 RX23W の使用する割り込みベクタを追加
		9	「2.8 コードサイズ」を変更
		12	「2.12 FIT モジュールの追加方法」を変更
		13-28	API 説明ページの「Reentrant」項目を削除
		41	「5.2 動作確認環境」 Rev.2.00 に対応する表を追加

Rev.	発行日	改訂内容	
		ページ	ポイント
2.00	2020.06.10	ﾌﾟﾛｸﾞﾗﾑ	以下を修正しました。 ■対象デバイス 全デバイス ■内容 複数の周辺機能から同時にアクセスされる可能性があるレジスタがあり、そのレジスタへの書き込みのアトミック性が確保できるように処理を変更。
2.01	2021.07.31	−	RX140 に対応
		7	「2.4 使用する割り込みベクタ」 RX140 の使用する割り込みベクタを追加
		9	「2.11 コールバック関数」を変更
		35	「表5.1　イベントリンク信号の定義一覧(1/3)」 LPT コンペアマッチ 1 を追加

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。