# RX ファミリ

## RTC 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)を使った RTC モジュールについて説明します。

本 FIT モジュールは、リアルタイムクロック(以降、RTC)の 24 時間モードとカレンダカウントモードをサポートします。本 FIT モジュールでサポートする機能には、日時、アラーム、カウント開始/停止、周期割り込み、クロック出力の設定が含まれます。RX230、RX231、RX23W、RX64M、RX65N、RX66N、RX71M、RX72M、RX72N では、時間キャプチャ機能もサポートします。また、アラーム割り込みか周期割り込みを使って、低消費電力状態から復帰できます。

以降、本モジュールを RTC FIT モジュールと称します。

### 対象デバイス

- RX110、RX111、RX113、RX130 グループ
- RX230、RX231、RX23W グループ
- RX64M グループ
- RX65N グループ
- RX66N グループ
- RX71M グループ
- RX72M グループ
- RX72N 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「6.1 動作確認環境」を参照してください。

### 関連アプリケーションノート

- RX ファミリ 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 ..... 3

2. API 情報 ..... 4
2.1 ハードウェアの要求 ..... 4
2.2 ハードウェアリソースの要求 ..... 4
2.2.1 RTC ..... 4
2.2.2 I/O ポート、MPC ..... 4
2.2.3 サブクロック発振器 ..... 4
2.3 ソフトウェアの要求 ..... 4
2.4 対応ツールチェーン ..... 4
2.5 使用する割り込みベクタ ..... 5
2.6 ヘッダファイル ..... 5
2.7 整数型 ..... 5
2.8 コンパイル時の設定 ..... 6
2.9 コードサイズ ..... 7
2.10 引数 ..... 8
2.11 コールバック関数 ..... 8
2.12 FIT モジュールの追加方法 ..... 9
2.13 for 文、while 文、do while 文について ..... 10

3. API 関数 ..... 11
3.1 概要 ..... 11
3.2 戻り値 ..... 11
3.3 R_RTC_Open() ..... 12
3.4 R_RTC_Close() ..... 15
3.5 R_RTC_Control() ..... 16
3.6 R_RTC_Read() ..... 21
3.7 R_RTC_GetVersion () ..... 23

4. 端子設定 ..... 24

5. デモプロジェクト ..... 25
5.1 rtc_demo_rskrx130 ..... 25
5.2 rtc_demo_rskrx231 ..... 25
5.3 rtc_demo_rskrx64m ..... 25
5.4 ワークスペースにデモを追加する ..... 25

6. 付録 ..... 26
6.1 動作確認環境 ..... 26
6.2 トラブルシューティング ..... 30

7. 参考ドキュメント ..... 31

# 1. 概要

本 FIT モジュールは RTC の 24 時間カレンダカウントモードに対応しています。ハードウェアの機能詳細については、ご使用の MCU のユーザーズマニュアル　ハードウェア編をご覧ください。

本 FIT モジュールは以下に示す RTC の機能に対応しています。

- 日時の設定
- カウントの開始／停止
- アラームの設定
- 周期割り込み
- クロック出力

RX230、RX231、RX64M、RX65N、RX66N、RX71M、RX72M、RX72N では、以下に示す 3 つの時間キャプチャイベント入力端子を使用できます。

- RTCIC0
- RTCIC1
- RTCIC2

RX23W では、以下に示す 2 つの時間キャプチャイベント入力端子を使用できます。

- RTCIC0
- RTCIC1

本 FIT モジュールでは、以下の機能はサポートしていません。

- 12 時間モード
- バイナリカウントモード
- 30 秒調整機能
- 時計誤差補正機能
- 桁上げ割り込み
- RTC のカウントソースにメインクロックを選択
  （RX64M、RX65N、RX66N、RX71M、RX72M、RX72N）

# 2. API 情報

本 FIT モジュールは、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。

- RTCc、RTCd、RTCe、RTCA

## 2.2 ハードウェアリソースの要求

ここでは、本 FIT モジュールが要求するハードウェアの周辺機能について説明します。特に記載がない場合、ここで説明するリソースは本 FIT モジュールが使用できるように、ユーザのプログラムでは使用しないでください。

### 2.2.1 RTC

本 FIT モジュールは、RTC を使用します。

### 2.2.2 I/O ポート、MPC

本 FIT モジュールは、クロック出力および時間キャプチャ機能が使用できます。これらの機能を使用する場合は、対応する端子の設定が必要です。

### 2.2.3 サブクロック発振器

RTC はサブクロックで動作します。本 FIT モジュールの API 関数を呼ぶ前に、サブクロックが発振および発振安定待ちが完了している必要があります。サブクロックの発振、および発振安定待ちの方法については、ご使用の MCU のユーザーズマニュアル ハードウェア編に従ってください。

## 2.3 ソフトウェアの要求

本 FIT モジュール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

## 2.4 対応ツールチェーン

本 FIT モジュールは「6.1 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。

## 2.5 使用する割り込みベクタ

R_RTC_Open 関数または R_RTC_Control 関数の引数に設定する値で、周期割り込みおよびアラーム割り込みを有効にできます。

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 |
|---|---|
| 対象デバイスすべて | ALM 割り込み（ベクタ番号: 92）<br>PRD 割り込み（ベクタ番号: 93） |

## 2.6 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_ rtc_rx_if.h に記載しています。

## 2.7 整数型

このドライバは 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。

## 2.8 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_rtc_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。

<table>
<tr><th colspan="2">コンフィギュレーションオプション（r_rtc_rx_config.h）</th></tr>
<tr><td>#define RTC_CFG_PARAM_CHECKING_ENABLE<br>※デフォルト値は “1”</td><td>本定義を“1”にするとパラメータチェック処理のコードを生成し、“0”にすると生成しません。<br>本定義を BSP_CFG_PARAM_CHECKING_ENABLE に設定するとシステムのデフォルト設定が使用されます。</td></tr>
<tr><td>#define RTC_CFG_CALCULATE_YDAY<br>※デフォルト値は “0”</td><td>本定義を“1”にすると、R_RTC_Read 関数を呼び出したとき、引数に指定した tm_t 構造体変数のメンバ tm_yday に 1 月 1 日からの日数を算出して格納します。 “0”にすると日数の算出をしません。</td></tr>
<tr><td>#define RTC_CFG_DRIVE_CAPACITY_STD<br>※デフォルトです。その他の設定値を以下に示します。<br><br>その他の設定:<br>//#define RTC_CFG_DRIVE_CAPACITY_LO<br>//#define RTC_CFG_DRIVE_CAPACITY_MD<br>//#define RTC_CFG_DRIVE_CAPACITY_HI</td><td>本定義はサブクロック発振器の駆動能力を設定します。サブクロック発振時に設定した駆動能力と同じ設定にしてください。
<table>
<tr><th rowspan="2">MCU</th><th colspan="4">駆動能力</th></tr>
<tr><th>低速<br>(LO)</th><th>中速<br>(MD)</th><th>高速<br>(HI)</th><th>標準<br>(STD)</th></tr>
<tr><td>RX11x</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>RX130<br>RX230<br>RX231<br>RX23W<br>RX64M<br>RX71M<br>RX72M<br>RX72N<br>RX65N<br>RX66N</td><td>X</td><td>-</td><td>-</td><td>X</td></tr>
</table>
X：設定可能 、- : 設定不可</td></tr>
</table>

## 2.9 コードサイズ

RX100 シリーズ、RX200 シリーズ、RX600 シリーズの主なデバイスについて、本モジュールで使用されるコード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、ビルド時のコンフィギュレーションオプション「2.8 コンパイル時の設定」によって決まります。掲載した値は、「2.4 対応ツールチェーン」の C コンパイラでコンパイルオプションがデフォルト時の参考値です。コンパイルオプションのデフォルトは最適化レベル：2、最適化のタイプ：サイズ優先、データ・エンディアン：リトルエンディアンです。コードサイズは 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。

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_rtc_rx rev2.78

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
(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 4.08.04.201902
(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 4.12.1
(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 |
| RX130 | ROM | 2,270 バイト | 1,891 バイト | 4,164 バイト | 3,508 バイト | 3,208 バイト | 2,665 バイト |
| | RAM | 8 バイト | | 8 バイト | | 14 バイト | |
| | スタック (注 1) | 112 バイト | | - | | 164 バイト | |
| RX231 | ROM | 2,707 バイト | 2,282 バイト | 5,020 バイト | 4,292 バイト | 3,887 バイト | 3,269 バイト |
| | RAM | 16 バイト | | 16 バイト | | 22 バイト | |
| | スタック (注 1) | 112 バイト | | - | | 164 バイト | |
| RX65N | ROM | 2,710 バイト | 2,285 バイト | 5,036 バイト | 4,276 バイト | 3,719 バイト | 3,090 バイト |
| | RAM | 16 バイト | | 16 バイト | | 20 バイト | |
| | スタック (注 1) | 124 バイト | | - | | 192 バイト | |

注1.　割り込み関数の最大使用スタックサイズを含みます。

## 2.10 引数

本 FIT モジュールの API で使用されるデータ構造体は、r_rtc_rx_if.h に記載されています。詳細は「3 API 関数」をご覧ください。

## 2.11 コールバック関数

本モジュールでは、周期割り込みまたはアラーム割り込み処理の内部で、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、「2.10 引数」に記載された構造体メンバ “p_callback” に、ユーザの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。

コールバック関数を使用しない場合、“p_callback”には‘FIT_NO_FUNC’を設定してください。

表 2.2　コールバック関数の引数一覧(enum rtc_cb_evt_t)

| 定数定義 | 説明 |
|---|---|
| RTC_EVT_ALARM | アラーム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 |
| RTC_EVT_PERIODIC | 周期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 |

```
コールバック関数使用例：
                :
rtc_init.p_callback = rtc_callback;             // コールバック関数名を設定
err = R_RTC_Open(&rtc_init, &init_time);        // RTC の初期設定
                :
void rtc_callback(void *p_args)
{
    rtc_cb_evt_t event;
    event = *(rtc_cb_evt_t *)p_args;
    if (event == RTC_EVT_PERIODIC)              // 周期割り込み
    {
        do_something_prd();
    }
    else if (event == RTC_EVT_ALARM)            // アラーム割り込み
    {
        do_something_alm();
    }
}
```

```
コールバック関数を使用しない場合：
                :
rtc_init.p_callback = FIT_NO_FUNC;      // ‘FIT_NO_FUNC’を設定
err = R_RTC_Open(&rtc_init, &init_time); // RTC の初期設定
                :
```

## 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、Smart Configurator を使用した(1)、(3)の追加方法を推奨しています。ただし、Smart Configurator 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 Smart Configurator を使用して FIT モジュールを追加する場合
$e^2$ studio の Smart Configurator 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「Renesas $e^2$ studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(2) $e^2$ studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
$e^2$ studio の FIT Configurator 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 Smart Configurator 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 Smart Configurator 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「Renesas $e^2$ studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」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。

## 2.13 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 概要

本 FIT モジュールの関数を以下に示します。

| 関数 | 説明 |
|---|---|
| R_RTC_Open() | RTC の初期設定を行い、現在日時をセットし、周期割り込みやクロック出力を設定してカウントを開始します。 |
| R_RTC_Close() | カウントを停止し、周期割り込み、アラーム割り込みを禁止します。 |
| R_RTC_Control() | 現在日時やアラーム日時を更新、時間キャプチャ機能(対応 MCU のみ)やその他の設定を行います。 |
| R_RTC_Read() | 現在の日時とアラームの日時を返します。 |
| R_RTC_GetVersion() | 本 FIT モジュールのバージョン番号を返します。 |

## 3.2 戻り値

以下に API 関数で戻り値として返すエラーコードの列挙型を示します。

```
typedef enum                    //RTC API の戻り値
{
  RTC_SUCCESS,
  RTC_ERR_ALREADY_OPEN,         //R_RTC_Open 関数が既に呼ばれています。
  RTC_ERR_NOT_OPENED,           //R_RTC_Open 関数が呼ばれていません。
  RTC_ERR_BAD_PARAM,            //存在しない、または無効なパラメータが指定されています。
  RTC_ERR_MISSING_CALLBACK,     //コールバック関数が設定されていません。

  RTC_ERR_TIME_FORMAT,          //不適切な時間フォーマット（フィールドが範囲外です。）
  RTC_ERR_NO_CAPTURE            //時間キャプチャイベントを検出していません。
} rtc_err_t;
```

## 3.3 R_RTC_Open()

RTC の初期設定を行い、現在時刻をセットし、関連する割り込みを設定して、カウントを開始します。

この関数は RTC FIT モジュールを初期化する関数です。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

```
rtc_err_t R_RTC_Open(rtc_init_t   *p_init,
                     tm_t         *p_current_time);
```

### Parameters

p_init

初期設定構造体へのポインタ（以下参照）

p_current_time

日時設定用構造体へのポインタ（以下参照）。現在時刻の設定に使用されます。

p_init に使用される初期設定構造体:

```
typedef struct
{
  rtc_cb_func_t  p_callback;          //コールバック関数のポインタを設定
  rtc_output_t   output_freq;         //クロック出力の周波数を設定
                                        (set_time = false 指定時、設定値無効)
  rtc_periodic_t periodic_freq;       //周期割り込みの周期を設定
  uint8_t        periodic_priority;   //周期割り込みの割り込み優先レベルを設定
                                        (0 ～ 15, 0=割り込み禁止)
  bool           set_time;            //RTC の初期化および日時設定を実行またはスキップ
                                        (true : 実行、false : スキップ)
} rtc_init_t;

typedef void (*rtc_cb_func_t)(void *p_args);

typedef enum e_rtc_output
{
  RTC_OUTPUT_OFF,
  RTC_OUTPUT_1_HZ,
  RTC_OUTPUT_64_HZ,
} rtc_output_t;

typedef enum e_rtc_periodic
{
  RTC_PERIODIC_OFF = 0,
  RTC_PERIODIC_256_HZ = 6,
  RTC_PERIODIC_128_HZ = 7,
  RTC_PERIODIC_64_HZ = 8,
  RTC_PERIODIC_32_HZ = 9,
  RTC_PERIODIC_16_HZ = 10,
  RTC_PERIODIC_8_HZ = 11,
  RTC_PERIODIC_4_HZ = 12,
  RTC_PERIODIC_2_HZ = 13,
  RTC_PERIODIC_1_HZ = 14,
  RTC_PERIODIC_2_SEC = 15,
} rtc_periodic_t;
```

p_current_time に使用される構造体:

```
Typedef struct
{
  int tm_sec;     //秒(0-59)
  int tm_min;     //分(0-59)
  int tm_hour;    //時(0-23)
  int tm_mday;    //日にち／月(1-31)
  int tm_mon;     //月(0-11, 0=1月)
  int tm_year;    //年(100-199, 100=2000年)
  int tm_wday;    //曜日(0-6, 0=日曜日)
  int tm_yday;    //日にち／年(0-365)：設定無効
                    (RTC_CFG_CALCULATE_YDAY が“1”の場合に使用されます)
  int tm_isdst;  //夏時間：設定無効
                    (“-1”が設定されます)
} tm_t;
```

### Return Values

*RTC_SUCCESS*

*RTC_ERR_ALREADY_OPEN*　　*/* R_RTC_Open 関数が既に呼ばれています。*/*

*RTC_ERR_BAD_PARAM*　　*/*存在しない、または無効なパラメータが指定されています。*/*

*RTC_ERR_MISSING_CALLBACK*　　*/*コールバック関数が設定されていません。*/*

*RTC_ERR_TIME_FORMAT*　　*/*不適切な時刻フォーマット（フィールドが範囲外です。）*/*

### Properties

r_rtc_rx_if.h にプロトタイプ宣言されています。

### Description

本関数は RTC の初期設定を行い、カウントを開始します。RTC の初期設定が完了し、正常にカウントが開始されると“RTC_SUCCESS”を返します。

構造体“rtc_init_t”のメンバ“set_time”に“true”を設定した場合、RTC を初期化し、引数“p_current_time”を使って日時を設定します。メンバ“set_time”が“false”の場合、RTC を初期化せず、引数“p_current_time”は無視されます。通常、コールドスタートの場合は“true”を、ウォームスタート（リセットなど）の場合は“false”を設定します。

引数“p_current_time”に使用する構造体“tm_t”は、C 言語の標準ライブラリで定義されています。コンパイラが対応していない場合、構造体“tm_t”は“r_rtc_rx_if.h”の定義を使用します。

**Example**

```
rtc_err_t err;
rtc_init_t rtc_init;

/* 現在の日時を 2015 年 8 月 31 日（月曜日） 11:59:20pm に設定 */
tm_t init_time =
{
  20,   //秒(0-59)
  59,   //分(0-59)
  23,   //時(0-23)
  31,   //日にち／月(1-31)
  7,    //月(0-11, 0=1 月)
  115, //年(100-199, 100=2000 年)
  1,    //曜日(0-6, 0=日曜日)
  0,    //日にち／年(0-365)：設定無効
  0,    //夏時間： 設定無効
};
rtc_init.output_freq = RTC_OUTPUT_1_HZ;     //1Hz の出力クロック生成
rtc_init.periodic_freq = RTC_PERIODIC_2_HZ; //周期割り込みを 0.5 秒ごとに生成
rtc_init.periodic_priority = 7;              //周期割り込みの優先レベルを 7 に設定
rtc_init.set_time = true;                    //RTC の初期設定および日時の設定を実施
rtc_init.p_callback = rtc_callback;          //コールバック関数を設定

err = R_RTC_Open(&rtc_init, &init_time);
```

**Special Notes:**

本関数を呼び出す前に、サブクロックが発振および発振安定待ちが完了している必要があります。サブクロックの発振、および発振安定待ちの方法については、ご使用の MCU のユーザーズマニュアル ハードウェア編に従ってください。

本関数は、コールドスタート/ウォームスタートに関わらず呼び出す必要があります。

クロック出力を使用する場合には以下の注意事項があります。

- R_RTC_Open 関数、または R_RTC_Control 関数でクロック出力の設定をした後、アプリケーションで端子の設定を行ってください。詳細は「4 端子設定」を参照してください。
- ウォームスタート時 (rtc_init_t->set_time = false ) は、R_RTC_Open 関数によるクロック出力の設定が無効になります。ウォームスタートでクロック出力を使用する場合は、R_RTC_Open 関数の呼び出し後に、R_RTC_Control 関数でクロック出力の設定を行ってください。

## 3.4 R_RTC_Close()

カウントを停止し、RTC をリセットして、すべての RTC 関連の割り込みを禁止します。

### Format

```
void R_RTC_Close (void);
```

### Parameters

なし

### Return Values

なし

### Properties

r_rtc_rx_if.h にプロトタイプ宣言されています。

### Description

カウントを停止し、RTC をリセットして、すべての RTC 関連の割り込みを禁止します。

### Example

```
rtc_err_t     err;
rtc_init_t    rtc_init;
tm_t          init_time;
    :
  err = R_RTC_Open(&rtc_init, &init_time);
    :
  R_RTC_Close();
```

### Special Notes:

なし

## 3.5 R_RTC_Control()

現在日時やアラーム日時を更新、時間キャプチャ機能(対応 MCU のみ)やその他の設定を行います。

### Format

```
rtc_err_t R_RTC_Control(rtc_cmd_t cmd,
                        void      *p_args);
```

### Parameters

cmd

処理するコマンド（下記の列挙型参照）

p_args

任意の引数の構造体へのポインタ（各コマンドに対する設定は Description を参照してください）

使用可能なコマンド:

```
typedef enum
{
  /* 全MCU */
  RTC_CMD_SET_OUTPUT,
  RTC_CMD_SET_PERIODIC,
  RTC_CMD_SET_CURRENT_TIME,
  RTC_CMD_SET_ALARM_TIME,
  RTC_CMD_ENABLE_ALARM,
  RTC_CMD_STOP_COUNTERS,
  RTC_CMD_START_COUNTERS,
  RTC_CMD_PARTIAL_RESET,

/* RX230、RX231、RX23W、RX64M、RX65N、RX66N、RX71M、RX72M、RX72N のみに適用*/
  RTC_CMD_CONFIG_CAPTURE,
  RTC_CMD_CHECK_PIN0_CAPTURE,
  RTC_CMD_CHECK_PIN1_CAPTURE,
  RTC_CMD_CHECK_PIN2_CAPTURE,
  RTC_CMD_DISABLE_CAPTURE
} rtc_cmd_t;
```

### Return Values

*RTC_SUCCESS*

*RTC_ERR_NOT_OPENED*　　*/* R_RTC_Open 関数が呼ばれていません。*/*

*RTC_ERR_BAD_PARAM*　　*/*存在しない、または無効なパラメータが指定されています。*/*

*RTC_ERR_MISSING CALLBACK*　　*/*コールバック関数が設定されていません。*/*

*RTC_ERR_TIME_FORMAT*　　*/*不適切な時間フォーマット（フィールドが範囲外です。）*/*

*RTC_ERR_NO_CAPTURE*　　*/*時間キャプチャイベントを検出していません。*/*

### Properties

r_rtc_rx_if.h にプロトタイプ宣言されています。

**Description**

本関数は、現在日時やアラーム日時を更新、時間キャプチャ機能(対応 MCU のみ)やその他の設定を行います。以下に各コマンドについて説明します。

**RTC_CMD_SET_OUTPUT:**

クロック出力の設定を変更することができます。設定には構造体 rtc_output_t を使用します。本コマンドによるクロック出力の設定中はカウントを一時的に停止します。以下にサンプルを示します。

```
rtc_output_t out_freq=RTC_OUTPUT_OFF;
err = R_RTC_Control(RTC_CMD_SET_OUTPUT, &out_freq);
```

**RTC_CMD_SET_PERIODIC:**

周期割り込みの生成時間を変更することができます。設定には構造体 rtc_periodic_cfg_t を使用します。以下にサンプルを示します。

```
rtc_periodic_cfg_t periodic;

periodic.frequency = RTC_PERIODIC_2_HZ; // 1/2 秒ごとに割り込みを生成
periodic.int_priority = 9;
err = R_RTC_Control(RTC_CMD_SET_PERIODIC, &periodic);
```

**RTC_CMD_SET_CURRENT_TIME:**

現在日時の設定を変更することができます。設定には構造体 tm_t を使用します。本コマンドによる日時の設定中はカウントを一時的に停止します。以下にサンプルを示します。

```
tm_t time;
  :
err = R_RTC_Control(RTC_CMD_SET_CURRENT_TIME, &time)
```

**RTC_CMD_SET_ALARM_TIME:**

アラーム日時を設定することができます。設定には構造体 tm_t を使用します。アラーム日時を再設定するときは、RTC_CMD_ENABLE_ALARM でアラーム機能を無効にしてから再設定してください。以下にサンプルを示します。

```
tm_t time;
  :
err = R_RTC_Control(RTC_CMD_SET_ALARM_TIME, &time);
```

**RTC_CMD_ENABLE_ALARM:**

アラーム日時と現在日時で、比較するフィールド(年、月、日、曜日など)を設定し、アラーム割り込みを有効にします。以下にサンプルを示します。

```
tm_t                time;
rtc_alarm_ctrl_t    alarm;

/* 毎月 1 日の午前 9:00 にアラームを作成 */
time.tm_sec = 0;    //秒(0-59)
time.tm_min = 0;    //分(0-59)
time.tm_hour = 9;   //時(0-23)
time.tm_mday = 1;   //日にち／月(1-31)
time.tm_mon = 0;    //月(0-11, 0=1 月)
time.tm_year = 100; //年(100-199, 100=2000 年)
time.tm_wday = 0;   //曜日(0-6, 0=日曜日)
err = R_RTC_Control(RTC_CMD_SET_ALARM_TIME, &time);

alarm.int_priority = 4; //アラーム割り込みの優先レベルを 4 に設定
alarm.sec = false;      //秒
alarm.min = false;      //分
alarm.hour = true;      //時    (true = 現在の時と比較する)
alarm.mday = true;      //日にち (true = 現在の日にちと比較する)
alarm.mon = false;      //月
alarm.year = false;     //年
alarm.wday = false;     //曜日
err = R_RTC_Control(RTC_CMD_ENABLE_ALARM, &alarm);
```

**RTC_CMD_STOP_COUNTERS:**

カウントを停止します。第二引数は NULL、または FIT_NO_PTR としてください。以下にサンプルを示します。

```
R_RTC_Control(RTC_CMD_STOP_COUNTERS, NULL);
```

**RTC_CMD_START_COUNTERS:**

RTC_CMD_STOP_COUNTERS で停止されたカウントを再開します。第二引数は NULL、または FIT_NO_PTR としてください。以下にサンプルを示します。

```
R_RTC_Control(RTC_CMD_START_COUNTERS, NULL);
```

**RTC_CMD_PARTIAL_RESET:**

クロック出力、アラームおよび時間キャプチャ関連レジスタをリセットするためのコマンドです（関連するレジスタの詳細は、ユーザーズマニュアル: ハードウェア編で RCR2.RESET ビットの説明をご覧ください）。第二引数は NULL、または FIT_NO_PTR としてください。以下にサンプルを示します。

```
R_RTC_Control(RTC_CMD_PARTIAL_RESET, NULL);
```

**RTC_CMD_CONFIG_CAPTURE:**

RTCIC0、RTCIC1、または RTCIC2 端子のイベント検出条件を設定します。設定には rtc_capture_cfg_t 構造体を使用します。以下にサンプルを示します。

```
rtc_capture_cfg_t capture;

capture.pin = RTC_PIN_0;
capture.edge = RTC_EDGE_RISING;
capture.filter = RTC_FILTER_OFF;
err = R_RTC_Control(RTC_CMD_CONFIG_CAPTURE, &capture);
```

**RTC_CMD_CHECK_PIN0_CAPTURE:**

**RTC_CMD_CHECK_PIN1_CAPTURE:**

**RTC_CMD_CHECK_PIN2_CAPTURE:**

キャプチャ端子の設定後、イベントの検出を確認するためにポーリングが必要です。キャプチャが行われた場合、第 2 引数で指定した引数にキャプチャ日時を格納し、RTC_SUCCESS を返します。キャプチャが行われていない場合は、RTC_ERR_NO_CAPTURE を返します。第 2 引数には構造体 tm_t を使用します。以下にサンプルを示します。

```
tm_t                time;
rtc_err_t           err;
rtc_capture_cfg_t   capture;
  :
err = R_RTC_Control(RTC_CMD_CONFIG_CAPTURE, &capture);

while(1)
{
  /* メイン処理 */
    :
  /* RTCIC0 端子でイベントを検出しているかどうかを確認 */
  if (R_RTC_Control(RTC_CMD_CHECK_PIN0_CAPTURE, &time) == RTC_SUCCESS)
  {
       /* ビジネス時間（9 時～5 時）外でイベントを検出した場合*/
       if ((time.tm_hour < 9) || (time.tm_hour > 17))
       {
          RED_LED = ON;
          write_flash(log_addr, sizeof(tm_t), &time);
          log_addr += sizeof(tm_t);
       }
  }
}
```

**RTC_CMD_DISABLE_CAPTURE:**

キャプチャ端子の設定を無効にすることができます。再度有効にするには、RTC_CMD_CONFIG_CAPTURE を使います。以下にサンプルを示します。

```
rtc_pin_t     pin=RTC_PIN_0;
err = R_RTC_Control(RTC_CMD_DISABLE_CAPTURE, &pin)
```

**Example**

```
/* 30 秒ごとにアラーム割り込み要求が発生するサンプル */

rtc_err_t err;
rtc_init_t rtc_init;
tm_t g_init_time={0, 0, 0, 1, 0, 100, 0, 0, 0};
rtc_alarm_ctrl_t alarm={4, false, false, false, false, false, false, false};
tm_t alm_time;

rtc_init.output_freq = RTC_OUTPUT_OFF;      // クロック出力しない
rtc_init.periodic_freq = RTC_PERIODIC_OFF;  // 周期割り込みを無効に設定
rtc_init.periodic_priority = 0;              // 周期割り込みの優先レベルを 0 に設定
rtc_init.set_time = true;                    //RTC の初期設定および日時の設定を実施
rtc_init.p_callback = rtc_callback;          // コールバック関数を設定

err = R_RTC_Open(&rtc_init, &g_init_time);

/* 秒の値が “30”になるとアラーム割り込み要求が発生するよう設定 */
alm_time = g_init_time;
alm_time.tm_sec = 30;
alm_time.int_priority = 7; //アラーム割り込みの優先レベルを 7 に設定
err = R_RTC_Control(RTC_CMD_SET_ALARM_TIME, &alm_time);

/* アラームの秒フィールドを有効にする */
alarm.sec = true;
err = R_RTC_Control(RTC_CMD_ENABLE_ALARM, &alarm);
  :
/* コールバック関数 */
void rtc_callback(void *p_args)
{
  rtc_err_t err;

  // 現在時刻を 0 に初期化。秒の値が“30”になるとアラーム割り込みが再度発生
  err = R_RTC_Control(RTC_CMD_SET_CURRENT_TIME, &g_init_time);

  // ここに処理を記載
}
```

**Special Notes:**

キャプチャ機能を使用する場合、R_RTC_Open 関数の呼び出し後、R_RTC_Control 関数で RTC_CMD_CONFIG_CAPTURE コマンドを実行する前に、アプリケーションで端子の設定を行ってください。詳細は「4 端子設定」を参照してください。

RTC_CMD_SET_OUTPUT コマンド、または RTC_CMD_SET_CURRENT_TIME コマンドを実行すると、設定中にカウントを一時的に停止します。

## 3.6 R_RTC_Read()

RTC で設定された現在日時とアラーム日時を返します。

### Format

```
rtc_err_t R_RTC_Read (tm_t * p_current_time,
                      tm_t * p_alarm_time);
```

### Parameters

p_current

RTC から現在の日時を読み込むためのポインタ。現在日時を読み飛ばす場合は、NULL か FIT_NO_PTR を設定してください。

p_alarm

RTC からアラーム日時を読み込むためのポインタ。アラーム日時を読み飛ばす場合は、NULL か FIT_NO_PTR を設定してください。

### Return Values

*RTC_SUCCESS*
*RTC_ERR_NOT_OPENED*　　/* R_RTC_Open 関数が呼ばれていません。*/

### Properties

r_rtc_rx_if.h にプロトタイプ宣言されています。

### Description

本関数は現在日時とアラーム日時を読み出します。

**Example**

```
tm_t      cur_time;
tm_t      alm_time;
rtc_err_t err;

err = R_RTC_Read(&cur_time, NULL);       // 現在日時のみ読み込み
err = R_RTC_Read(NULL, &alm_time);       // アラーム日時のみ読み込み
err = R_RTC_Read(&cur_time, &alm_time); // 現在日時とアラーム日時を読み込み
```

**Special Notes:**

リセット、ディープソフトウェアスタンバイモード、ソフトウェアスタンバイモード、またはバッテリバックアップ機能からの復帰後に本関数で現在日時を読み出すときは、カウントが開始している状態（RCR2.START ビット = 1）で、1/128 秒の待ち時間が必要です。

本関数では、現在日時の読み出し中に RTC カウンタの桁上げが発生した場合、現在日時の読み直しを行います。桁上げの確認には、桁上げ割り込みの割り込みステータスフラグ(IR ビット)を使用しています。そのため、桁上げ割り込みを許可(RCR1.CIE ビット = 1)に設定しています。ユーザプログラムでこのステータスフラグをクリアしないでください。

## 3.7 R_RTC_GetVersion ()

この関数は本 FIT モジュールのバージョン番号を返します。

### Format

```
uint32_t R_RTC_GetVersion(void)
```

### Parameters

なし

### Return Values

*バージョン番号*

### Properties

r_rtc_rx_if.h にプロトタイプ宣言されています。

### Description

この関数は本 FIT モジュールのバージョン番号を返します。最上位の 2 バイトがメジャーバージョン番号で、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。

### Example

```
uint32_t version;
version = R_RTC_GetVersion();
```

### Special Notes:

なし。

## 4. 端子設定

RTC FIT モジュールを使用するためには、マルチファンクションピンコントローラ（MPC）で周辺機能の入出力信号を端子に割り付ける（以下、端子設定と称す）必要があります。

RTCOUT 端子は以下に従って設定してください。

- コールドスタート時 (rtc_init_t->set_time = true ) は、R_RTC_Open 関数、または R_RTC_Control 関数でクロック出力の設定をした後に RTCOUT の端子設定を行ってください。
- ウォームスタート時 (rtc_init_t->set_time = false ) は、R_RTC_Open 関数によるクロック出力の設定が無効になります。R_RTC_Open 関数の呼び出し後に、R_RTC_Control 関数でクロック出力の設定をした後に RTCOUT の端子設定を行ってください。

RTCICn(n=0～2)端子は R_RTC_Open 関数を呼び出した後、R_RTC_Control 関数で RTC_CMD_CONFIG_CAPTURE コマンドを実行する前に設定してください。

e$^2$ studio の場合は「FIT Configurator」または「Smart Configurator」の端子設定機能を使用することができます。FIT Configurator、Smart Configurator の端子設定機能を使用すると、端子設定画面で選択したオプションに応じて、ソースファイルが出力されます。そのソースファイルで定義された関数を呼び出すことにより端子を設定できます。詳細は表 4.1 を参照してください。

表 4.1 FIT コンフィグレータが出力する関数一覧

| 使用マイコン | 出力される関数名 | 備考 |
|---|---|---|
| 全デバイス共通 | R_RTC_PinSet() | RX100 シリーズには RTCICn(n=0～2)の設定が出力されません。 |

# 5. デモプロジェクト

デモプロジェクトはスタンドアロンプログラムです。デモプロジェクトには、FIT モジュールとそのモジュールが依存するモジュール（例：r_bsp）を使用する main()関数が含まれます。

## 5.1 rtc_demo_rskrx130

説明:

RSKRX130（FIT モジュール “r_rtc_rx”）向けの RX130 リアルタイムクロック（RTCc）のシンプルなデモです。デモでは RTC FIT モジュールの API（r_rtc_rx_if.h に記載）を使って、任意の日時にリアルタイムクロックを初期設定し、2 秒ごとに周期割り込みを発生させます。割り込み処理では現在の日時を読み込み、グローバル変数に格納します。格納した値は、Main()関数によって、デバッグコンソールに書き出されます。2 秒ごとの周期割り込みで LED0 がトグルします。

設定と実行:

1. サンプルコードをコンパイルしてダウンロードします。
2. ソフトウェアを実行します。PC が Main で停止した場合、F8 を押して再開します。
3. ブレークポイントを設定し、グローバル変数を確認します。

対応ボード:

- RSKRX130

## 5.2 rtc_demo_rskrx231

説明:

RSKRX231（FIT モジュール “r_rtc_rx”）向けの RX231 リアルタイムクロック（RTCe）のシンプルなデモです。デモの内容は RX130 と同じです。

対応ボード:

- RSKRX231

## 5.3 rtc_demo_rskrx64m

説明:

RSKRX64M（FIT モジュール “r_rtc_rx”）向けの RX64M リアルタイムクロック（RTCd）のシンプルなデモです。デモの内容は RX130 と同じです。

対応ボード:

- RSKRX64M

## 5.4 ワークスペースにデモを追加する

デモプロジェクトは、e$^2$ studio のインストールディレクトリ内の FITDemos サブディレクトリにあります。ワークスペースにデモプロジェクトを追加するには、「ファイル」→「インポート」を選択し、「インポート」ダイアログから「一般」の「既存プロジェクトをワークスペースへ」を選択して「次へ」ボタンをクリックします。「インポート」ダイアログで「アーカイブ・ファイルの選択」ラジオボタンを選択し、「参照」ボタンをクリックして FITDemos サブディレクトリを開き、使用するデモの zip ファイルを選択して「完了」をクリックします。

# 6. 付録

## 6.1 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 6.1 動作確認環境 (Rev.2.41)

| 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 V4.2.0.012 |
| 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 V2.04.01 |
| | コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加<br>-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 | Rev2.41 |
| 使用ボード | Renesas Starter Kit for RX130 (型名：RTK5005130SxxxxxBE) |

表 6.2 動作確認環境 (Rev.2.50)

| 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 V5.0.1.005 |
| 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 V2.05.00 |
| | 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<br>-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 | Rev2.50 |
| 使用ボード | Renesas Starter Kit+ for RX65N (型名：RTK500565NSxxxxxBE) |

表 6.3 動作確認環境 (Rev.2.70)

| 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.XXX |
| 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<br>-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 | Rev2.70 |
| 使用ボード | Renesas Starter Kit+ for RX65N-2MB (型名：RTK50565N2SxxxxxBE)<br>Renesas Starter Kit for RX130-512KB (型名：RTK5051308SxxxxxBE) |

表 6.4　動作確認環境 (Rev.2.71)

| 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.XXX |
| 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<br>-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 | Rev2.71 |
| 使用ボード | Renesas Starter Kit+ for RX65N-2MB (型名：RTK50565N2SxxxxxBE)<br>Renesas Starter Kit for RX130-512KB (型名：RTK5051308SxxxxxBE)<br>Renesas Starter Kit for RX231 (型名：R0K505231SxxxBE)<br>Renesas Starter Kit for RX113 (型名：R0K505113SxxxBE) |

表 6.5　動作確認環境 (Rev.2.72)

| 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 V6.1.0.XXX |
| 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<br>-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 | Rev2.72 |
| 使用ボード | Renesas Starter Kit+ for RX64M (型名：R0K50564MSxxxBE)<br>Renesas Starter Kit for RX130 (型名：RTK5005130SxxxxxBE)<br>Renesas Starter Kit for RX231 (型名：R0K505231SxxxBE) |

表 6.6　動作確認環境 (Rev.2.73)

| 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 V7.1.0 |
| 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 V3.00.00 |
| | 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<br>-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 | Rev2.73 |

表 6.7　動作確認環境 (Rev.2.74)

| 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 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<br>-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 | Rev2.74 |

表 6.8　動作確認環境 (Rev.2.75)

| 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 V7.3.0<br>IAR Embedded Workbench for Renesas RX 4.10.01 |
| 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 V.3.01.00<br>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<br>-lang = c99 |
| | GCC for Renesas RX 4.08.04.201803<br>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<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 4.10.01<br>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 | Rev2.75 |
| 使用ボード | Renesas Starter Kit+ for RX65N (型名：RTK500565Nxxxxxx) |

表 6.9　動作確認環境 (Rev.2.76)

| 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 V7.2.0 |
| 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 V.3.01.00 |
| | 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<br>-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 | Rev2.76 |
| 使用ボード | Renesas Solution Starter Kit for RX23W (型名：RTK5523Wxxxxxxxxxx) |

表 6.10　動作確認環境 (Rev.2.77)

| 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 V7.4.0<br>IAR Embedded Workbench for Renesas RX 4.12.01 |
| 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 V.3.01.00<br>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<br>-lang = c99 |
| | GCC for Renesas RX 4.08.04.201902<br>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<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 4.12.01<br>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 | Rev2.77 |
| 使用ボード | Renesas Starter Kit+ for RX72M (型名：RTK5572Mxxxxxxxxxx) |

表 6.11　動作確認環境 (Rev.2.78)

| 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 V7.4.0<br>IAR Embedded Workbench for Renesas RX 4.12.01 |
| 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 V.3.01.00<br>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<br>-lang = c99 |
| | GCC for Renesas RX 4.08.04.201902<br>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<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 4.12.01<br>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 | Rev2.78 |
| 使用ボード | Renesas Starter Kit+ for RX72N (型名：RTK5572Nxxxxxxxxxx) |

## 6.2 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、ビルド実行すると以下のエラーが発生します。
「ERROR - A drive capacity #define must be uncommented in r_rtc_rx_config.h」
「ERROR - Only one drive capacity #define may be uncommented in r_rtc_rx_config.h」
「ERROR - RTC_CFG_DRIVE_CAPACITY_MD in r_rtc_rx_config.h is invalid selection for MCU.」

A : “r_rtc_rx_config.h” ファイルの設定値が間違っている可能性があります。“r_rtc_rx_config.h” ファイルを確認して正しい値を設定してください。詳細は「2.8 コンパイル時の設定」を参照してください。

(3) Q : RTCOUT 端子からクロックが出力されません。

A : 正しく端子設定が行われていない可能性があります。本 FIT モジュールを使用する場合は端子設定が必要です。詳細は「4 端子設定」を参照してください。

(4) Q : RTCICn(n=0～2)端子にエッジを入力してもイベントが検出されません。

A : イベントの検出条件や端子設定が正しく行われていない可能性があります。「3.5 R_RTC_Control()」の RTC_CMD_CONFIG_CAPTURE コマンドの設定を確認してください。また、RTCICn(n=0～2)端子が汎用入力ポートの設定になっていることを確認してください。

(5) Q : R_RTC_Open 関数を呼び出しても関数内で無限ループし、カウントが開始されません。

A : サブクロックが正常に発振できていない可能性があります。ユーザーズマニュアル ハードウェア編の設定手順に従い、R_RTC_Open 関数を呼び出す前に、サブクロックが発振していることを確認してください。

(6) Q : ウォームスタートすると常にカウンタが初期化されます。

A: ウォームスタートで R_RTC_Open 関数を呼び出すとき、rtc_init_t 構造体の set_time に“false” が設定されていることを確認してください。また、コールドスタート/ウォームスタート判別フラグ(RSTSR1.CWSF)を使用して処理を分岐している場合、コールドスタートの処理内で、このフラグが “1” (ウォームスタート)に設定できていることを確認してください。

## 7. 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 CC-RX コンパイラ ユーザーズマニュアル（R20UT3248）
（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の対応について

本モジュールは以下のテクニカルアップデートの内容を反映しています。

- 対応しているテクニカルアップデートはありません。

## 改訂記録

| Rev. | 発行日 | 改訂内容 | |
|---|---|---|---|
| | | ページ | ポイント |
| 2.60 | 2017.3.31 | — | 初版発行 |
| | | 4<br><br><br><br><br><br>5<br><br><br><br><br>8<br>9<br>10<br><br>11,12,13<br><br><br><br><br><br><br><br><br><br><br><br>15,16,17<br>,18,19<br><br><br><br><br><br><br><br><br><br><br>20,21<br><br><br><br><br><br><br>23 | ・「2.2 ハ―ドウェアリソースの要求」<br>2.2.2 I/O ポート、MPC の説明文を見直し<br>2.2.3 サブクロック発振器 のサブクロックに関する説明を見直し<br>・「制限事項」章の削除<br>・「2.4 対応ツールチェーン」の説明文を見直し<br>・「2.5 使用する割り込みベクタ」の追加<br>・「2.6 ヘッダファイル」の説明文を見直し<br>・「2.7 整数型」の説明文を見直し<br>・「2.8 コンパイル時の設定」の説明文を見直し<br>・「2.11 コールバック関数」の追加<br>・「2.12 FIT モジュールの追加方法」の説明文を見直し<br>・「3.1 概要」の説明文を見直し<br>・「3.2 戻り値」の説明文を見直し<br>・「3.3 R_RTC_Open()」<br>Parameters<br>構造体の説明文を見直し<br>Description<br>説明文を見直し<br>Example<br>サンプルコードを見直し<br>Special Notes<br>サブクロックに関する説明を見直し<br>コールバック関数の記載を 2.11 コールバック関数に移動<br>・「3.5 R_RTC_Control()」<br>Parameters<br>構造体の説明文を見直し<br>Description<br>各コマンドに関する説明文とサンプルコードを見直し<br>Example<br>サンプルコードを見直し<br>Special Notes<br>キャプチャ端子に関する説明文を見直し<br>・「3.6 R_RTC_Read()」<br>Parameters<br>構造体の説明文を削除<br>Description<br>説明文を見直し<br>・「4. 端子設定」<br>説明文を見直し |

| Rev. | 発行日 | 改訂内容 | |
|---|---|---|---|
| | | ページ | ポイント |
| 2.70 | 2017.7.31 | -<br>-<br><br>1<br><br>4<br><br>7<br>24<br>24<br>26 | ・RX65N-2MB、RX130-512KB に対応<br>・RX210、RX631、RX63N に関連する CGC FIT モジュールの公開が取り消されたことに伴い、対応デバイスから RX210、RX631、RX63N を削除<br>・「関連アプリケーションノート」に Renesas e2 studio スマート・コンフィグレータユーザーガイド(R20AN0451) を追加<br>・「2.1 ハ―ドウェアの要求」から RTCa(RX210)、RTCb(RX631、RX63N)を削除<br>・「2.9 コードサイズ」の ROM サイズを見直し<br>・「デモプロジェクト」章の削除<br>・「5. 付録」の追加<br>・「6.参考ドキュメント」の追加 |
| 2.71 | 2017.9.20 | 10<br><br>12<br><br>15<br><br><br><br><br><br><br><br><br>18<br><br><br>20<br><br>22<br><br>24 | ・「3.1 R_RTC_Open()」<br>Parameters<br>output_freq メンバに関する説明を追加<br>Special Notes<br>説明を追加<br>・「3.5 R_RTC_Control()」<br>Description<br>RTC_CMD_SET_OUTPUT コマンド<br>説明追加<br>RTC_CMD_SET_CURRENT_TIME コマンド<br>説明追加<br>RTC_CMD_SET_ALARM_TIME コマンド<br>説明追加<br>Special Notes<br>説明を追加<br>・「3.6 R_RTC_Read()」<br>Special Notes<br>説明追加<br>・「4.端子設定」<br>説明追加<br>・「5.1 動作確認環境」章に表 5.4 を追加 |
| | | プログラム | ・以下の不具合を修正。<br><br>**<u>現在日時の読み出し処理不具合</u>**<br>■内容<br>R_RTC_Read 関数を使って現在日時の読み出し中に RTC カウンタの桁上げが発生した場合、誤った日時が読み出される。<br>(桁上げが発生した場合に、現在日時の読み直しを行う仕組みとなっているが、不具合により桁上げを検出できずに現在日時の読み直しが行われない)<br><br>例) 時刻が 0 時 0 分 59 秒のとき、秒の読み出し直後に桁上げが発生した場合<br>実時刻 0:01:00 、読み出し時刻 0:01:59 (59 秒の誤差) |

| Rev. | 発行日 | 改訂内容 | |
|---|---|---|---|
| | | ページ | ポイント |
| 2.71 | 2017.9.20 | プログラム | ■発生条件<br>RTC カウンタを読み出し中に RTC カウンタの桁上げが発生したとき。<br><br>■対策<br>RTC FIT モジュール Rev2.71 以降をご使用ください。<br>本修正により、以下の定義値を変更しています。<br><br>RTC_INT_ENABLE (0x05) → (0x07)<br><br>これにより、桁上げ割り込み許可ビット(RCR1.CIE)が許可になり、現在日時の読み出し中の桁上げを検出して、読み直しを行うようになります。<br><br><u>**ウォームスタート時のカウント一時停止不具合**</u><br>■内容<br>R_RTC_Open 関数の引数“rtc_init_t”のメンバ“set_time”に“false”を設定すると、R_RTC_Open 関数の処理の過程でカウントが一時停止します。<br>(ウォームスタート設定にした場合に、カウントが一時的に停止します。)<br><br>■発生条件<br>R_RTC_Open 関数の引数“rtc_init_t”のメンバ“set_time”に“false”を設定したとき。<br><br>■対策<br>RTC FIT モジュール Rev2.71 以降をご使用ください。<br>以下の修正を行っています。<br><br>R_RTC_Open 関数内で呼び出している rtc_set_output 関数を、コールドスタート時のみ実施するルーチン内に移動。<br><br>これにより、ウォームスタート時にカウントが一時停止しなくなります。<br><br>・制限事項の追加<br><u>**ウォームスタート時のカウント一時停止不具合**</u><br>の不具合改修により、ウォームスタート時の R_RTC_Open 関数におけるクロック出力の設定は無効です。 |
| 2.72 | 2017.12.14 | 4<br><br><br>23<br><br>25 | ・「2.4. 対応ツールチェーン」<br>下記内容を修正<br>「6.1 動作確認環境」→「6.1 動作確認環境」<br>・「5.デモプロジェクト」<br>説明追加<br>・「6.1 動作確認環境」章に表 6.5 を追加 |

| Rev. | 発行日 | 改訂内容 | |
|---|---|---|---|
| | | ページ | ポイント |
| 2.73 | 2018.12.03 | 25 | 6.1 動作確認環境<br>表 6.6 動作確認環境 (Rev.2.73)を追加。 |
| | | プログラム | FIT モジュールのサンプルプログラムをダウンロードするためのアプリケーションノートのドキュメント番号を xml ファイルに追加。 |
| 2.74 | 2019.02.01 | 25 | 6.1 動作確認環境<br>表 6.7 動作確認環境 (Rev.2.74)を追加。 |
| | | プログラム | 機能関連<br>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<br>■内容<br>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。 |
| 2.75 | 2019.05.20 | — | 以下のコンパイラに対応<br>・GCC for Renesas RX<br>・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」 R01AN1723、R01AN1826、R20AN0451 を削除 |
| | | 4 | 「2.3 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 |
| | | 6 | 「2.9 コードサイズ」を更新 |
| | | 21 | 3.7 R_RTC_GetVersion に関数のインライン展開を削除 |
| | | 26 | 6.1 動作確認環境 に、表 6.8 動作確認環境 (Ver.2.75)を追加 |
| 2.76 | 2019.06.20 | 1 | 「対象デバイス」に RX23W グループを追加<br>「関連ドキュメント」R01AN1833 を削除 |
| | | 3 | 「1 概要」に、RX23W の時間キャプチャイベント入力端子の情報を追加 |
| | | 6 | 「2.8 コンパイル時の設定」サブクロック発振器の駆動能力表に RX23W を追加 |
| | | 7 | 「2.9 コードサイズ」を更新 |
| | | 9 | 「2.13 for 文、while 文、do while 文について」を追加 |
| | | 28 | 6.1 動作確認環境 に、表 6.9 動作確認環境 (Ver.2.76)を追加 |
| 2.77 | 2019.07.30 | 1 | 「対象デバイス」に RX72M グループを追加 |
| | | 3 | 「1 概要」に、RX72M の時間キャプチャイベント入力端子の情報を追加 |
| | | 6 | 「2.8 コンパイル時の設定」サブクロック発振器の駆動能力表に RX72M を追加 |
| | | 7 | 「2.9 コードサイズ」を更新 |
| | | 13—23 | 「3 API 関数」各 API の Reentrant を削除 |
| | | 28 | 6.1 動作確認環境 に、表 6.10 動作確認環境 (Ver.2.77)を追加 |

| Rev. | 発行日 | 改訂内容 | |
|---|---|---|---|
| | | ページ | ポイント |
| 2.78 | 2019.11.22 | 1 | 「対象デバイス」に RX66N、RX72N グループを追加 |
| | | 3 | 「1 概要」に、RX66N、RX72N の時間キャプチャイベント入力端子の情報を追加 |
| | | 6 | 「2.8 コンパイル時の設定」サブクロック発振器の駆動能力表に RX66N、RX72N を追加 |
| | | 7 | 「2.9 コードサイズ」を更新 |
| | | 29 | 6.1 動作確認環境　に、表 6.11 動作確認環境 (Ver.2.78)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、ノイズ輻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

## 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。

## 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/

© 2019 Renesas Electronics Corporation. All rights reserved.