RX ファミリ

CMTW モジュール Firmware Integration Technology

要旨

本モジュールは CMTW を 2 ユニット持つ RX ファミリで使用可能です。本ドキュメントでは、それらのユニットをチャネルと称します。各チャネルには、コンペアマッチ（CM）が 1 つ、アウトプットコンペア（OC0、OC1）が 2 つ、インプットキャプチャ（IC0、IC1）が 2 つあります。コンペアマッチは汎用タイマで、周期タイマとして使用できます。アウトプットコンペアは、タイマの設定に従って波形を出力する機能です。インプットキャプチャは、端子にイベント信号が入力されたときにタイマカウンタをキャプチャする機能です。CMTW タイマは 32 ビット幅で、数百ナノ秒～数時間までの設定が可能です。

本ドキュメントは CMTW FIT モジュール API について説明します。CMTW FIT モジュールをユーザアプリケーションと使用するための情報として、ソフトウェアアーキテクチャ、システムインタフェース、および使用に関する詳細を提供します。

対象デバイス

- RX64M グループ
- RX651 グループ、RX65N グループ
- RX71M グループ
- RX72M 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。

ターゲットコンパイラ

- ルネサスエレクトロニクス製 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環境に関する詳細な内容は、セクション「6.1 動作確認環境 」を参照してください。

目次

1. 概要

本モジュールは、RX の周辺機能である CMTW を設定するための API を提供します。

本モジュールでは、CMTW 周辺機能の割り込み処理をサポートし、コールバック関数を使ってユーザアプリケーションへの通知を行います。

CMTW は 2 チャネルあるため、各 API 関数で使用する CMTW チャネルを指定してください。タイマの初期設定は、R_CMTW_Open() API 関数で行います。この API は、指定された CMTW チャネルを設定用パラメータに基づいて初期化し、必要に応じて割り込みを有効にします。タイマのイベントが発生したとき、指定したコールバック関数を呼び出します。コールバック関数の呼び出しは、CMTW の割り込み処理で行われるため、システム内の他の処理に時間を使えるように、コールバック関数はできるだけ早く処理を完了してください。また、設定用パラメータにてワンショット動作を指定することもできます。

初期設定完了後、R_CMTW_Control()API を使用して、タイマを開始、停止、再開、リスタートすることができます。CMTW 動作が必要でなくなれば、R_CMTW_Close() API で CMTW チャネルを無効にし、終了させることで電力消費を低減できます。

図 1 に CMTW FIT モジュールを使ったプロジェクトの例を示します。

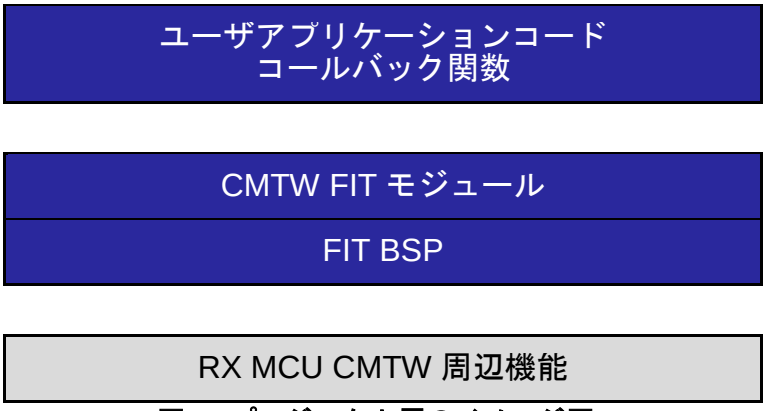

図 1 プロジェクト層のイメージ図

1.1 CMTW Firmware Integration Technology (FIT)モジュールを使用する

CMTW FIT モジュールの使用目的は、CMTW タイマの設定と制御を簡単に行えるようにすることです。ユーザアプリケーションでは、タイマイベント発生時に実行されるコールバック関数を任意で割り当てることができます。

CMTW FIT モジュールをプロジェクトに追加後、r_cmtw_rx_config.h ファイルを変更する必要があります。コンフィギュレーションオプションの詳細についてはセクション 2.9 を参照してください。

CMTW FIT モジュールを使用する前に、CMTW 周辺機能で使用する入出力端子を正しく設定する必要があります。本モジュールでは端子の初期設定は行いませんので、CMTW の API を呼び出す前に設定する必要があります。この設定は、汎用端子の初期設定処理の一部として、システムのスタートアップ時に行われるのが一般的です。端子の設定には、GPIO FIT モジュールと MPC FIT モジュールを使用できます。表 1 に CMTW 周辺機能での入出力端子の割り当てを示します。

表 1 CMTW の端子設定

CMTW の入出力端子の割り当て		
チャネル 0	アウトプットコンペア 0 (TOC0)	PC7
	アウトプットコンペア 1 (TOC1)	PE7
	インプットキャプチャ 0 (TIC0)	PC6
	インプットキャプチャ 1 (TIC1)	PE6
チャネル 1	アウトプットコンペア 0 (TOC2)	PD3
	アウトプットコンペア 1 (TOC3)	PE3
	インプットキャプチャ 0 (TIC2)	PD2
	インプットキャプチャ 1 (TIC3)	PE2

1.2 割り込み

CMTW FIT モジュールでは CMTW のすべての割り込みに対して割り込み処理を提供しているため、割り込みベクタを設定する必要はありません。R_CMTW_Open() API に提供されるユーザ設定のパラメータに応じて、割り込みのみ、あるいはコールバック関数を実行するアクションが取られます。CMTW を DTC や DMAC など、他の周辺機能で使用する場合、割り込みのみのアクションが有用です。ただし、本 FIT モジュールでは、CMTW 以外の周辺機能の設定は行いません。

コールバックは、CMTW タイマイベントを通知するために使用されます。必要に応じて、ユーザ提供のコールバック関数が CMTW ISR によって呼び出されます。コールバック関数では、ISR に応じてユーザコードが実行されます。コールバック関数は割り込み処理の中で処理されるため、その間はその他の割り込みが禁止されます。そのため、システムで発生し得るその他の割り込みが失われないように、コールバック関数の処理をできるだけ早く完了することを推奨します。

割り込みが必要ない場合、タイマのみを実行するアクションを使用できます。この場合、割り込みを生成しないので、CPU 時間を消費することなく、アウトプットコンペア波形を生成することができます。

1.3 コールバック関数

1.3.1 コールバック関数のプロトタイプ宣言例

```
void my_cmtw_callback(void *pdata);
```

1.3.2 pdata 引数を型変換する

ISR コードは r_cmtw_rx_if.h ファイルで定義された構造体へのポインタを使用し、ユーザコールバック関数にイベント情報を渡します。ポインタで示される構造体は cmtw_callback_data_t です。FIT のコールバック関数は void 型のポインタを取るため、CMTW 割り込み処理で提供される情報にアクセスできるように、ポインタを型変換（cmtw_callback_data_t *）する必要があります。割り込みのデータ構造体および型はセクション 2.11.1 で定義されます。

例：

```
void my_cmtw_callback(void *pdata)
{
cmtw_callback_data_t *p_cb_data = (cmtw_callback_data_t *)pdata;
....
cb_data.channel = p_cb_data->channel;
cb_data.event = p_cb_data->event;
cb_data.count = p_cb_data->count;
...
}
```

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。

- CMTW

2.2 ハードウェアリソースの要求

ここでは、本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。

- 特に要求はありません。

2.3 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）v5.20 以降のバージョン

本 FIT モジュールの API 関数呼び出し前に、外部で必要な入出力端子が正しく初期設定されていることを前提としています。

2.4 制限事項

本 FIT モジュールは CMTW のみに適用されます。他の周辺機能が CMTW と関連付けられている場合、その周辺機能に関する設定は外部で行ってください。セクション 2.9 でロックの使用と BSP への依存についてご確認ください。

2.5 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。

2.6　割り込みベクタ

CMTW 割り込みは、**R_CMTW_Open** 関数を実行すると有効化されます。
表 2.1　CMTW FIT モジュールで使用する割り込みベクタには、CMTW FIT モジュールで使用する割り込みベクタの一覧を記載しています。

表 2.1　CMTW FIT モジュールで使用する割り込みベクタ

デバイス	割り込みベクタ
RX64M RX651, RX65N RX71M, RX72M	● CMWI0 割り込み[チャネル 0]（ベクタ番号：30） ● CMWI1 割り込み[チャネル 1]（ベクタ番号：31） ● IC0I0 割り込み[チャネル 0]（ベクタ番号：168）*1 ● IC1I0 割り込み[チャネル 0]（ベクタ番号：169）*1 ● OC0I0 割り込み[チャネル 0]（ベクタ番号：170）*1 ● OC1I0 割り込み[チャネル 0]（ベクタ番号：171）*1 ● IC0I1 割り込み[チャネル 1]（ベクタ番号：172）*1 ● IC1I1 割り込み[チャネル 1]（ベクタ番号：173）*1 ● OC0I1 割り込み[チャネル 1]（ベクタ番号：174）*1 ● OC1I1 割り込み[チャネル 1]（ベクタ番号：175）*1
注 1：ソフトウェアで設定可能な割り込み B の割り込みベクタ番号は、ボードサポートパッケージ FIT モジュール（BSP モジュール）で指定したデフォルト値を示しています。	

2.7 ヘッダファイル

すべての API 呼び出しとそれらをサポートするインタフェース定義は r_cmtw_rx_if.h に記載しています。

r_cmtw_rx_config.h ファイルで、ビルド時に設定可能なコンフィギュレーションオプションを選択あるいは定義できます。

上記 2 ファイルはいずれもユーザアプリケーションにインクルードする必要があります。

2.8 整数型

コードをわかりやすく、また移植が容易に行えるように、本プロジェクトでは ANSI C99 (Exact width integer types (固定幅の整数型)) 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。

2.9 コンパイル時の設定

ビルド時に設定可能なコンフィギュレーションオプションは r_cmtw_rx_config.h ファイルに含まれます。下表に各設定の概要を示します。

表 2 CMTW FIT モジュールのコンフィギュレーションオプション

コンフィギュレーションオプション（r_cmtw _rx_config.h）	
CMTW_CFG_PARAM_CHECKING_ENABLE	CMTW の API 関数のパラメータチェック処理をコード出力するか、しないかを選択します。システムでコードサイズを小さくして処理速度を上げることが要求される場合、パラメータチェックを無効にできます。 デフォルトでは、システム全体に対して BSP_CFG_PARAM_CHECKING_ENABLE マクロが使用される設定になっています。本マクロを再定義することによって、CMTW での設定をオーバーライドできます。本定義を 1 に設定するとパラメータチェック処理のコードを生成し、0 に設定すると生成しません。
CMTW_CGF_REQUIRE_LOCK	API 関数の呼び出しが同時に発生しないように、API 関数をロックします。RTOS 環境で、API 関数への複数アクセスを回避します。 初期設定では、ロックは有効に設定されています。システムでマルチタスクが行われず、API 関数のインスタンスが複数実行される可能性がない場合、本定義を 0 に設定してロックを無効にすることで、コードの格納領域を確保できます。
CWTW_CFG_IPR_CM_CH0	チャネル 0 に対して、コンペアマッチ割り込みの優先レベルを定義します。有効値は 1～15 です。
CMTW_CFG_IPR_OC0_CH0	チャネル 0 に対して、アウトプットコンペア 0 割り込みの優先レベルを定義します。有効値は 1～15 です。
CMTW_CFG_IPR_OC1_CH0	チャネル 0 に対して、アウトプットコンペア 1 割り込みの優先レベルを定義します。有効値は 1～15 です。
CMTW_CFG_IPR_IC0_CH0	チャネル 0 に対して、インプットキャプチャ 0 割り込みの優先レベルを定義します。有効値は 1～15 です。
CMTW_CFG_IPR_IC1_CH0	チャネル 0 に対して、インプットキャプチャ 1 割り込みの優先レベルを定義します。有効値は 1～15 です。
CWTW_CFG_IPR_CM_CH1	チャネル 1 に対して、コンペアマッチ割り込みの優先レベルを定義します。有効値は 1～15 です。
CMTW_CFG_IPR_OC0_CH1	チャネル 1 に対して、アウトプットコンペア 0 割り込みの優先レベルを定義します。有効値は 1～15 です。
CMTW_CFG_IPR_OC1_CH1	チャネル 1 に対して、アウトプットコンペア 1 割り込みの優先レベルを定義します。有効値は 1～15 です。
CMTW_CFG_IPR_IC0_CH1	チャネル 1 に対して、インプットキャプチャ 0 割り込みの優先レベルを定義します。有効値は 1～15 です。
CMTW_CFG_IPR_IC1_CH1	チャネル 1 に対して、インプットキャプチャ 1 割り込みの優先レベルを定義します。有効値は 1～15 です。

2.10 コードサイズ

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「2.9 コンパイル時の設定」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。掲載した値は、「2.5 対応ツールチェーン」の C コンパイラのコンパイルオプションで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とコンパイルオプションによって異なります。

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						備考
			ルネサス製コンパイラ		GCC		IAR コンパイラ		
			パラメータチェックあり、ロック有効	パラメータチェックなし、ロック有効	パラメータチェック処理あり、ロック必須	パラメータチェック処理なし、ロック必須	パラメータチェック処理あり、ロック必須	パラメータチェック処理なし、ロック必須	
RX64M	ROM	1 チャネル使用	2025 バイト	1798 バイト	3036 バイト	2708 バイト	4746 バイト	4439 バイト	
		2 チャネル使用	2522 バイト	2295 バイト	3632 バイト	3304 バイト	5199 バイト	4892 バイト	
	RAM	1 チャネル使用	32 バイト	32 バイト	0 バイト	0 バイト	16 バイト	16 バイト	
		2 チャネル使用	64 バイト	64 バイト	0 バイト	0 バイト	32 バイト	32 バイト	
	最大使用スタックサイズ		64 バイト	64 バイト	-	-	176 バイト	176 バイト	
RX65N	ROM	1 チャネル使用	2025 バイト	1798 バイト	3036 バイト	2708 バイト	4746 バイト	4439 バイト	
		2 チャネル使用	2522 バイト	2295 バイト	3632 バイト	3304 バイト	5199 バイト	4892 バイト	
	RAM	1 チャネル使用	32 バイト	32 バイト	0 バイト	0 バイト	16 バイト	16 バイト	
		2 チャネル使用	64 バイト	64 バイト	0 バイト	0 バイト	32 バイト	32 バイト	
	最大使用スタックサイズ		64 バイト	64 バイト	-	-	176 バイト	176 バイト	

RX71M	ROM	1 チャネル使用	2025 バイト	1664 バイト	3036 バイト	2708 バイト	4746 バイト	4439 バイト	
		2 チャネル使用	2522 バイト	2161 バイト	3632 バイト	3304 バイト	5199 バイト	4892 バイト	
	RAM	1 チャネル使用	32 バイト	32 バイト	0 バイト	0 バイト	16 バイト	16 バイト	
		2 チャネル使用	64 バイト	64 バイト	0 バイト	0 バイト	32 バイト	32 バイト	
	最大使用スタックサイズ		64 バイト	64 バイト	-	-	176 バイト	176 バイト	
RX72M	ROM	1 チャネル使用	2033 バイト	1806 バイト	3356 バイト	2972 バイト	2917 バイト	2606 バイト	
		2 チャネル使用	2530 バイト	2303 バイト	3952 バイト	3568 バイト	3371 バイト	3060 バイト	
	RAM	1 チャネル使用	32 バイト	32 バイト	0 バイト	0 バイト	16 バイト	16 バイト	
		2 チャネル使用	64 バイト	64 バイト	0 バイト	0 バイト	32 バイト	32 バイト	
	最大使用スタックサイズ		120 バイト	120 バイト	-	-	220 バイト	220 バイト	

2.11 API データ型

本モジュールの API で使用されるデータ構造体について説明します。

2.11.1 データ型

API 関数で使用するパラメータの多くは、enum 型で定義しています。これは型チェックを行い、エラーを減少させるためです。使用可能な値は、r_cmtw_rx_if.h ファイルに定義されます。

以下にデータ型の定義を示します。

```
/* チャネル番号 */
typedef enum
{
  CMTW_CHANNEL_0 = 0,
  CMTW_CHANNEL_1,
  CMTW_CHANNEL_MAX,
} cmtw_channel_t;

/* 時間ベース */
typedef enum
{
  CMTW_TIME_NSEC = 0,
  CMTW_TIME_USEC,
  CMTW_TIME_MSEC,
  CMTW_TIME_SEC,
  CMTW_TIME_MAX,
} cmtw_time_unit_t;

/* PCLK 分周設定 */
typedef enum
{
  CMTW_CLK_DIV_8 = 0,   // PCLK/8
  CMTW_CLK_DIV_32,      // PCLK/32
  CMTW_CLK_DIV_128,     // PCLK/128
  CMTW_CLK_DIV_512,     // PCLK/512
  CMTW_CLK_DIV_MAX,
} cmtw_clock_divisor_t;
```

```
/* カウンタクリア要因 */
typedef enum
{
  CMTW_CLR_CMT = 0,
  CMTW_CLR_DISABLED = 1,
  CMTW_CLR_IC0 = 4,
  CMTW_CLR_IC1 = 5,
  CMTW_CLR_OC0 = 6,
  CMTW_CLR_OC1 = 7,
  CMTW_CLR_MAX,
} cmtw_clear_source_t;

/* CMTW API のアクション*/
typedef enum
{
  CMTW_ACTION_NONE = 0x00,        // アクションなし
  CMTW_ACTION_TIMER = 0x01,       // タイマを実行、割り込みなし
  CMTW_ACTION_INTERRUPT = 0x02,   // 割り込み要求を生成
  CMTW_ACTION_CALLBACK = 0x04,    // 割り込み要求を生成し、ユーザ定義のコールバックを実行
  CMTW_ACTION_ONESHOT = 0x08,     // 割り込みを生成、動作は一回のみでチャネルを終了
} cmtw_actions_t;

/* 出力端子の状態 */
typedef enum
{
  CMTW_OUTPUT_RETAIN = 0,  // 端子の状態を変更しない
  CMTW_OUTPUT_LO_TOGGLE,   // Low 出力して、その後トグル
  CMTW_OUTPUT_HI_TOGGLE,   // High 出力して、その後トグル
CMTW_OUTPUT_MAX,
} cmtw_output_states_t;

/* キャプチャする入力端子のエッジ */
typedef enum
{
  CMTW_EDGE_RISING = 0,    // 立ち上がりエッジをキャプチャ
  CMTW_EDGE_FALLING,       // 立ち下がりエッジをキャプチャ
  CMTW_EDGE_ANY,           // 立ち上がり、立ち下がりの両エッジをキャプチャ
  CMTW_EDGE_MAX,
} cmtw_edge_states_t;

/* Control 関数のコマンドコード */
typedef enum
{
  CMTW_CMD_START,       // タイマを開始
  CMTW_CMD_RESUME,      // 開始と同じ動作
  CMTW_CMD_STOP,        // タイマを停止
  CMTW_CMD_RESTART,     // カウンタを 0 にして、タイマを開始
  CMTW_CMD_MAX,         // 無効なコマンド
} cmtw_cmd_t;

/* Open 関数の CM 設定 */
typedef struct
{
  uint32_t time;
  cmtw_actions_t actions;
} cmtw_cm_settings_t;
```

```
/* Open 関数の OC 設定 */
typedef struct
{
  uint32_t time;
  cmtw_actions_t actions;
  cmtw_output_states_t output;
} cmtw_oc_settings_t;


/* Open 関数の IC 設定 */
typedef struct
{
  cmtw_actions_t actions;
  cmtw_edge_states_t edge;
} cmtw_ic_settings_t;

/* Open 関数のチャネル設定 */
typedef struct
{
  cmtw_time_unit_t time_unit;
  cmtw_clock_divisor_t clock_divisor;
  cmtw_clear_source_t clear_source;
  cmtw_cm_settings_t cm_timer;
  cmtw_oc_settings_t oc_timer_0;
  cmtw_oc_settings_t oc_timer_1;
  cmtw_ic_settings_t ic_timer_0;
  cmtw_ic_settings_t ic_timer_1;
} cmtw_channel_settings_t;

/* コールバック関数のイベント */
typedef enum
{
  CMTW_EVENT_CM = 0,    //コンペアマッチ
  CMTW_EVENT_IC0,       //インプットキャプチャ 0
  CMTW_EVENT_IC1,       //インプットキャプチャ 1
  CMTW_EVENT_OC0,       //アウトプットコンペア 0
  CMTW_EVENT_OC1,       //アウトプットコンペア 1
} cmtw_event_t;

/* コールバック関数のデータ構造体 */
typedef struct
{
  cmtw_channel_t channel;  //イベントチャネル番号
  cmtw_event_t event;      //イベントタイプ
  uint32_t count;          //イベント発生時のタイマカウンタ
} cmtw_callback_data_t;
```

2.12 戻り値

API 関数の戻り値を示します。戻り値は、r_cmtw_rx_if.h で定義されています。

```
/* CMTW 関数の戻り値 */
typedef enum
{
  CMTW_SUCCESS = 0,
  CMTW_ERR_BAD_CHAN,            // 無効なチャネル番号
  CMTW_ERR_CH_NOT_ENABLED,      // ユーザによってチャネルは無効に設定されています。
  CMTW_ERR_CH_NOT_OPENED,       // チャネルはオープン状態ではありません。
  CMTW_ERR_CH_NOT_CLOSED,       // チャネルはオープンにされたままです。
  CMTW_ERR_CH_NOT_RUNNIG,       // タイマ停止中に停止コマンドを受信しました。
  CMTW_ERR_CH_NOT_STOPPED,      // タイマ動作中に開始コマンドを受信しました。
  CMTW_ERR_UNKNOWN_CMD,         // コントロールコマンドが認識できません。
  CMTW_ERR_INVALID_ARG,         // パラメータに対して無効な引数です。
  CMTW_ERR_NULL_PTR,            // 引数に Null ポインタが指定されました。Null ポインタは無効です。
  CMTW_ERR_LOCK,                // ロックに失敗しました。
  CMTW_ERR_OUT_OF_RANGE,        // 算出されたカウント値は範囲外です。
} cmtw_err_t;
```

2.13 FIT モジュールをプロジェクトに追加する方法

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) e² studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e² 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 e² 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) e² studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e² 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 e2 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 e² 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。

2.14　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

本モジュールには以下の関数が含まれます。

関数	説明
R_CMTW_Open()	指定された CMTW チャネルを開始し、使用する CMTW レジスタを初期化し、必要に応じて割り込みを有効にします。割り込みイベントに応じてコールバック関数のポインタを取ります。処理を正常に完了すると、チャネルはオープン状態になります。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。
R_CMTW_Control()	オープン状態の CMTW チャネルを開始、停止、再開、リスタートします。
R_ CMTW_Close()	指定された CMTW チャネルを無効にして終了します。チャネルはクローズ状態になります。
R_CMTW_GetVersion()	本モジュールのバージョン番号を返します。

3.2 R_CMTW_Open ()

指定された CMTW チャネルを開始し、使用する CMTW レジスタを初期化し、必要に応じて割り込みを有効にします。割り込みイベントに応じてコールバック関数のポインタを取ります。処理を正常に完了すると、チャネルはオープン状態になります。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

```
cmtw_err_t R_CMTW_Open(cmtw_channel_t          channel,
                       cmtw_channel_settings_t    *pconfig,
                       void (* const pcallback)(void *pdata));
```

Parameters

channel

初期化する CMTW チャネル番号

pconfig

CMTW チャネル設定用データ構造体へのポインタ

pcallback

割り込みから呼び出されるユーザ関数へのポインタ

Return Values

CMTW_SUCCESS	/*成功; チャネルが初期化されました。*/
CMTW_ERR_BAD_CHAN	/*無効なチャネル番号 */
CMTW_ERR_CH_NOT_ENABLED	/*ユーザによってチャネルは無効に設定されています。*/
CMTW_ERR_CH_NOT_CLOSED	/*チャネルは動作中です。先に R_CMTW_Close()を 実行してください。*/
CMTW_ERR_INVALID_ARG	/* *pconfig* 構造体の要素に無効な値が含まれます。*/
CMTW_ERR_OUT_OF_RANGE	/*算出したカウント値は範囲外です。*/
CMTW_ERR_NULL_PTR	/* *pconfig* または *pcallback* が null です。*/
CMTW_ERR_LOCK	/* ロックできませんでした。チャネルはビジー状態です。*/

Properties

r_cmtw_rx_if.h にプロトタイプ宣言されています。

Description

本関数は CMTW チャネルを設定します。本関数が完了すると、CMTW チャネルは初期化され、R_CMTW_Control()関数を呼び出して開始することができます。他の CMTW API 関数を呼び出す前に、本関数を呼び出す必要があります。本関数が正常に完了すると、選択された CMTW チャネルはオープン状態になります。その後は、R_CMTW_Close()関数で当該チャネルを終了することなく、同一のチャネルに対して本関数を呼び出さないでください。

Example 1

ここではチャネル 0 を設定し、500ms ごとにコンペアマッチを実施し、コールバック関数を使用する例を示します。クリア要因には CMTW_CLR_CMT を設定し、500ms ごとにコンペアマッチが発生するように設定します。設定可能なクリア要因イベントはいくつかありますが、1 つのチャネルに対して設定できる要因は 1 つです。不適切なクリア要因が選択された場合、タイマカウンタはクリアされません。以下の例では、コンペアマッチイベント発生ごとにコールバック関数 cb を呼び出す例を示します。

```
cmtw_err_t rc;
cmtw_channel_settings_t ch;

/* cmtw_channel_settings_t 構造体のフィールドをすべてクリア */
memset(&ch, 0, sizeof(ch));

/* 500ms 周期でコンペアマッチによるコールバック関数の呼び出し設定*/
ch.time_unit = CMTW_TIME_MSEC;
ch.clock_divisor = CMTW_CLK_DIV_8;
ch.clear_source = CMTW_CLR_CMT;
ch.cm_timer.time = 500;
ch.cm_timer.actions = CMTW_ACTION_CALLBACK;
rc = R_CMTW_Open(CMTW_CHANNEL_0, &ch, &cb);

if (CMTW_SUCCESS != rc)
{
/* エラー処理 */
}
```

Example 2

ここではチャネル 0 を使い、アウトプットコンペア 0（OC0）およびアウトプットコンペア 1（OC1）の動作を設定する例を示します。OC0 アクションは CMTW_ACTION_TIMER に設定します。このイベントに対して割り込みは生成されません。PC7 端子が TOC0 に設定されている場合、50ms ごとに反転出力します。同様に PE7 端子を TOC1 に設定している場合、50ms ごとに出力が反転します。これは、クリア要因が CMTW_CLR_OC1 のためで、タイマカウンタは OC1 イベント（50ms に設定）ごとにクリアされます。OC0 イベントは 10ms で発生しますので、同じ期間を共有する位相の異なるイベントが 2 つ生成されることになります。OC1 アクションにコールバックを設定しているため、コールバック関数 cb が、チャネル 0 の OC1 ISR から呼び出されます。

この例では、クリア要因が CMTW_CLR_OC0 に設定されると、TOC0 が 10ms ごとに切り替えられることになります。タイマカウンタが 10ms でクリアされるので、50ms までカウントされることはなく、OC1 イベントは発生しません。

```
cmtw_err_t rc;
cmtw_channel_settings_t ch;

/* cmtw_channel_settings_t 構造体のフィールドをすべてクリア */
memset(&ch, 0, sizeof(ch));

/* TOC0 と TOC1 を切り替えて、50ms ごとのアウトプットコンペア設定 */
ch.time_unit = CMTW_TIME_MSEC;
ch.clock_divisor = CMTW_CLK_DIV_8;
ch.clear_source = CMTW_CLR_OC1;
ch.oc_timer_0.time = 10;
ch.oc_timer_0.actions = CMTW_ACTION_TIMER;
ch.oc_timer_0.output = CMTW_OUTPUT_HI_TOGGLE;
ch.oc_timer_1.time = 50;
ch.oc_timer_1.actions = CMTW_ACTION_CALLBACK;
ch.oc_timer_1.output = CMTW_OUTPUT_HI_TOGGLE;
rc = R_CMTW_Open(CMTW_CHANNEL_0, &ch, &cb);

if (CMTW_SUCCESS != rc)
{
/* エラー処理 */
}
```

Example 3

ここではチャネル 1 を使い、インプットキャプチャ 0（IC0）を設定する例を示します。正しく動作させるためには、CMTW の API 関数を呼び出す前に、PD2 端子を TIC2 に割り当てる必要があります。クリア要因に CMTW_CLR_DISABLED を設定しているため、TIC2 イベント（いずれのエッジにも設定される）発生時、タイマカウンタはクリアされません。コールバック関数 cb は、エッジの変化検出時にチャネル 1 の IC0 ISR から呼び出されます。チャネル番号、ISR イベント、イベント発生時のタイマの値（timer.tick カウント）がコールバック関数に渡されます。連続する TIC2 イベント間のカウント数が必要な場合、コールバック関数の 1 つ前のカウント値から現カウント値を減算することで求められます。

```
cmtw_err_t rc;
cmtw_channel_settings_t ch;

/* cmtw_channel_settings_t 構造体のフィールドをすべてクリア */
memset(&ch, 0, sizeof(ch));

/*インプットキャプチャ設定、クリアなし */
ch.time_unit = CMTW_TIME_USEC;
ch.clock_divisor = CMTW_CLK_DIV_128;
ch.clear_source = CMTW_CLR_DISABLED;
ch.ic_timer_0.actions = CMTW_ACTION_CALLBACK;
ch.ic_timer_0.edge = CMTW_EDGE_ANY;
rc = R_CMTW_Open(CMTW_CHANNEL_1, &ch, &cb);

if (CMTW_SUCCESS != rc)
{
/* エラー処理 */
}
```

Example 4

ここでは Example 3 のコールバック関数の例を示します。void ポインタの pdata は、データに正しくアクセスするため、“（cmtw_callback_data_t *）”に型変換しています。データ型の情報は r_cmtw_rx_if.h ファイルにあります。前回と今回のカウント値はグローバル変数に格納されます。連続する TIC2 イベント間の時間はこれらの値の差となります。タイマの tick 情報（例: 周辺クロックおよび CMTW クロック分周器）を利用できるので、実時間を容易に算出できます。

```
/* グローバル変数 */
uint32_t g_previous_count;
uint32_t g_current_count;
uint32_t g_delta_count;

void cb(void *pdata)
{
cmtw_callback_data_t cb_data;
cmtw_callback_data_t *p_cb_data = (cmtw_callback_data_t *)pdata;

cb_data.channel = p_cb_data->channel;
cb_data.event = p_cb_data->event;
cb_data.count = p_cb_data->count;

g_previous_count = g_current_count;
g_current_count = cb_data.count;
g_delta_count = g_current_count - g_previous_count;
}
```

3.3 R_CMTW_Control()

オープン状態の CMTW チャネルを開始、停止、再開、リスタートします。

Format

```
cmtw_err_t R_CMTW_Control(cmtw_channel_t channel,
                          cmtw_cmd_t cmd);
```

Parameters

channel

制御する CMTW チャネル番号

cmd

コマンドコードの列挙:

CMTW_CMD_START: タイマの開始

CMTW_CMD_RESUME: 開始と同じ動作

CMTW_CMD_STOP: タイマの停止

CMTW_CMD_RESTART: カウンタをゼロに設定して、タイマを起動

Return Values

CMTW_SUCCESS	/*成功; コマンドが実行されました。*/
CMTW_ERR_BAD_CHAN	/*無効なチャネル番号*/
CMTW_ERR_CH_NOT_ENABLED	/*ユーザによってチャネルは無効に設定されています。*/
CMTW_ERR_CH_NOT_OPENED	/*チャネルはクローズ状態です。先に R_CMTW_Open()を実行してください。*/
CMTW_ERR_CH_NOT_RUNNIG	/*チャネルは開始されていません。R_CMTW_Control()を実行して開始してください。*/
CMTW_ERR_CH_NOT_STOPPED	/*チャネルは動作中です。R_CMTW_Control()を実行して停止してください。*/
CMTW_ERR_UNKNOWN_CMD	/*無効なコマンド*/
CMTW_ERR_LOCK	/*ロックできませんでした。チャネルはビジー状態です。*/

Properties

r_cmtw_rx_if.h にプロトタイプ宣言されています。

Description

本関数はオープン状態のチャネルを開始、または再開します。正常に完了すると、選択された CMTW チャネルの状態が動作中に設定されます。停止コマンドは動作中のチャネルを停止します。この状態では、タイマのレジスタは停止コマンド実行直前の値を保持します。タイマは開始、または再開コマンドによって動作を継続します。リスタートコマンドはタイマカウンタをクリアして動作を再開します。

Example 1

ここではオープン状態のチャネル 0 を開始する例を示します。

```
/* タイマを起動 */
rc = R_CMTW_Open(CMTW_CHANNEL_0, &ch, &cb);

if (CMTW_SUCCESS != rc)
{
/* エラー処理 */
}

/* タイマを開始 */
rc = R_CMTW_Control(CMTW_CHANNEL_0, CMTW_CMD_START);

if (CMTW_SUCCESS != rc)
{
/* エラー処理 */
}
```

Example 2

ここでは動作中のチャネル 0 を停止する例を示します。

```
/* タイマを起動 */
rc = R_CMTW_Open(CMTW_CHANNEL_0, &ch, &cb);

if (CMTW_SUCCESS != rc)
{
/* エラー処理 */
}

/* タイマを開始 */
rc = R_CMTW_Control(CMTW_CHANNEL_0, CMTW_CMD_START);

if (CMTW_SUCCESS != rc)
{
/* エラー処理 */
}

/* タイマを停止 */
rc = R_CMTW_Control(CMTW_CHANNEL_0, CMTW_CMD_STOP);

if (CMTW_SUCCESS != rc)
{
/* エラー処理 */
}
```

3.4 R_CMTW_Close()

指定された CMTW チャネルを無効にして終了します。チャネルはクローズ状態になります。

Format

```
cmtw_err_t R_CMTW_Close(cmtw_channel_t channel);
```

Parameters

channel

終了する CMTW チャネル番号

Return Values

CMTW_SUCCESS	/*成功; チャネルが終了されました。*/
CMTW_ERR_BAD_CHAN	/*無効なチャネル番号*/
CMTW_ERR_CH_NOT_ENABLED	/*ユーザによってチャネルは無効に設定されています。*/
CMTW_ERR_CH_NOT_OPENED	/*チャネルはクローズ状態です。先に R_CMTW_Open()を実行してください。*/
CMTW_ERR_LOCK	/*ロックできませんでした。チャネルはビジー状態です。*/

Properties

r_cmtw_rx_if.h にプロトタイプ宣言されています。

Description

本関数はオープン状態または動作中の CMTW チャネルを停止して、無効にします。正常に完了すると、選択された CMTW チャネルは停止状態になり、チャネルを無効にすることで電力消費を低減します。当該チャネルは R_CMTW_Open()関数で再度開始されるまで使用できません。

Example 1

ここではチャネル 0 を終了する例を示します。

```
/* タイマを終了 */
rc = R_CMTW_Close(CMTW_CHANNEL_0);
if (CMTW_SUCCESS != rc)
{
/* エラー処理 */
}
```

3.5 R_CMTW_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_CMTW_GetVersion(void);
```

Parameters

なし

Return Values

メジャーバージョンとマイナーバージョンからなる 32 ビット値で示されるバージョン番号

Properties

r_cmtw_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。

Example

ここでは本関数の使用例を示します。

```
/* バージョン番号を取り出して、文字列に変換 */
uint32_t version, major, minor;
char version_str[11];

version = R_CMTW_GetVersion();

major = (version >> 16)&0xf;
minor = version & 0xff;

sprintf(version_str, "CMTW v%1.0hu.%2.2hu", major, minor);
```

Special Notes:

なし

4. 端子の設定

CMTW FIT モジュールを使用するには、周辺機能の出力信号をマルチファンクションピンコントローラ（MPC）で持つ端子に割り当てます。本書では、端子の割り当てを「端子設定」と呼びます。R_CMTW_Open() 関数を呼び出す前に、端子設定を行なってください。

e² studio で端子設定を行なう場合は、FIT コンフィグレータまたはスマート・コンフィグレータの端子設定機能を使用できます。端子設定機能を使用する場合、FIT コンフィグレータまたはスマート・コンフィグレータの端子設定画面で選択したオプションに従い、ソースファイルが生成されます。端子は、そのソースファイルで定義された関数を呼び出すと設定されます。

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。デモプロジェクトの標準的な命名規則は、<module>_demo_<board>となり、<module>は周辺の略語(例: s12ad、CMT、SCI）、<board>は標準 RSK（例: rskrx113）です。例えば、RSKRX113 用の s12ad FIT モジュールのデモプロジェクトは s12ad_demo_rskrx113 となります。同様にエクスポートされた.zip ファイルは <module>_demo_<board>.zip となります。例えば、zip 形式のエクスポート/インポートされたファイルは s12ad_demo_rskrx113.zip となります。

5.1 cmtw_demo_rskrx64m

説明:

RSKRX64M（FIT モジュール “r_cmtw_rx”）向けの RX64M コンペアマッチタイマ W（CMTW）のシンプルなデモです。デモでは、チャネル 0 を使って、500ms ごとのコンペアマッチ動作と割り込みコールバックを設定し、チャネル 1 を使って、1sec ごとのコンペアマッチ動作と割り込みコールバックを設定します。チャネル 0 によってコンペアマッチ割り込みが生成されると、割り込みコールバック関数が呼び出されて、500ms ごとに LED0 がトグルします。チャネル 1 によってコンペアマッチ割り込みが生成されると、割り込みコールバック関数が呼び出され、1sec ごとに LED1 がトグルします。

設定と実行:

1. 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

- RSKRX64M

5.2 cmtw_demo_rskrx71m

説明:

RSKRX71M（FIT モジュール “r_cmtw_rx”）向けの RX71M コンペアマッチタイマ W（CMTW）のシンプルなデモです。デモでは、チャネル 0 を使って、500ms ごとのコンペアマッチ動作と割り込みコールバックを設定し、チャネル 1 を使って、1sec ごとのコンペアマッチ動作と割り込みコールバックを設定します。チャネル 0 によってコンペアマッチ割り込みが生成されると、割り込みコールバック関数が呼び出されて、500ms ごとに LED0 がトグルします。チャネル 1 によってコンペアマッチ割り込みが生成されると、割り込みコールバック関数が呼び出され、1sec ごとに LED1 がトグルします。

設定と実行:

1. 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

- RSKRX71M

5.3 cmtw_demo_rskrx65n

説明:

RSKRX65N（FIT モジュール “r_cmtw_rx”）向けの RX65N コンペアマッチタイマ W（CMTW）のシンプルなデモです。デモでは、チャネル 0 を使って、500ms ごとのコンペアマッチ動作と割り込みコールバックを設定し、チャネル 1 を使って、1sec ごとのコンペアマッチ動作と割り込みコールバックを設定します。チャネル 0 によってコンペアマッチ割り込みが生成されると、割り込みコールバック関数が呼び出されて、500ms ごとに LED0 がトグルします。チャネル 1 によってコンペアマッチ割り込みが生成されると、割り込みコールバック関数が呼び出され、1sec ごとに LED1 がトグルします。

設定と実行:

1. 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

- RSKRX65N

5.4 cmtw_demo_rskrx65n_2m

説明:

RSKRX65N-2MB（FIT モジュール “r_cmtw_rx”）向けの RX65N-2MB コンペアマッチタイマ W（CMTW）のシンプルなデモです。デモでは、チャネル 0 を使って、500ms ごとのコンペアマッチ動作と割り込みコールバックを設定し、チャネル 1 を使って、1sec ごとのコンペアマッチ動作と割り込みコールバックを設定します。チャネル 0 によってコンペアマッチ割り込みが生成されると、割り込みコールバック関数が呼び出されて、500ms ごとに LED0 がトグルします。チャネル 1 によってコンペアマッチ割り込みが生成されると、割り込みコールバック関数が呼び出され、1sec ごとに LED1 がトグルします。

設定と実行:

1. 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

- RSKRX65N-2MB

5.5 ワークスペースへのデモ追加

デモプロジェクトは、本アプリケーションノートで提供されるファイルの 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「終了」をクリックします。

5.6 デモのダウンロード方法

デモプロジェクトは、RX Driver Package には同梱されていません。デモプロジェクトを使用する場合は、個別に各 FIT モジュールをダウンロードする必要があります。「スマートブラウザ」の「アプリケーションノート」タブから、本アプリケーションノートを右クリックして「サンプル・コード（ダウンロード）」を選択することにより、ダウンロードできます。

6. 付録

6.1 動作確認環境

このセクションでは、CMTW FIT モジュールの動作確認用の環境について説明します。

表 6.1 動作確認環境 (Rev.2.10)

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 V.7.5.0 IAR Embedded Workbench for Renesas RX 4.12.1
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package 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
	GCC for Renesas RX 4.8.4.201902 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 リンクオプション：「Optimize size (サイズ最適化) (-Os)」を使用する場合、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 -Wl,--no-gc-sections これは、FIT 周辺機器モジュール内で宣言されている割り込み関数をリンカが誤って破棄（discard）することを回避（work around）するための対策です。
	IAR C/C++ Compiler for Renesas RX version 4.12.1 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	Rev.2.10
使用ボード	Renesas Starter Kit+ for RX72M（型名：RTK5572Mxxxxxxxxxx）

表 6.2 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 V.7.4.0 IAR Embedded Workbench for Renesas RX 4.10.1
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package 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
	GCC for Renesas RX 4.8.4.201803 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 リンクオプション：「Optimize size (サイズ最適化) (-Os)」を使用する場合、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 -Wl,--no-gc-sections これは、FIT 周辺機器モジュール内で宣言されている割り込み関数をリンカが誤って破棄（discard）することを回避（work around）するための対策です。
	IAR C/C++ Compiler for Renesas RX version 4.10.1 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	Rev.2.00
使用ボード	Renesas Starter Kit+ for RX65N-2MB（型名：RTK50565Nxxxxxxxxx）

表 6.3　動作確認環境（Rev.1.32）

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 -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	Rev1.32
使用ボード	Renesas Starter Kit+ for RX65N-2MB（型名：RTK50565N2CxxxxxBR）

表 6.4　動作確認環境（Rev.1.31）

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	Rev1.31
使用ボード	Renesas Starter Kit+ for RX65N-2MB（型名：RTK50565N2CxxxxxBR）

表 6.5　動作確認環境（Rev.1.30）

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	Rev1.30
使用ボード	Renesas Starter Kit+ for RX65N-2MB（型名：RTK50565N2CxxxxxBR）

6.2 トラブルシューティング

(1) Q：プロジェクトに FIT モジュールを追加し、プロジェクトをビルドしました。次のエラーが発生しました。「ソースファイル「platform.h」を開くことができません。」

A：FIT モジュールがプロジェクトに対して正しく追加されていない可能性があります。次の文書で、FIT モジュールを追加した方法が正しいかどうか確認してください。

- 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)』
- e² 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 e2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)』

FIT モジュールを追加する場合、ボードサポートパッケージの 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。『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：プロジェクトに FIT モジュールを追加し、プロジェクトをビルドしました。次のエラーが発生しました。「現在の r_cmtw_rx モジュールはこの MCU をサポートしていません。」

A：追加した FIT モジュールは、現在のプロジェクトで選択されている対象デバイスをサポートしていない可能性があります。追加した FIT モジュールがサポートしているデバイスを確認してください。

(3) Q：プロジェクトに FIT モジュールを追加し、プロジェクトをビルドしました。構成の設定が誤っている場合に対応するエラーが発生しました。

A：「r_cmtw_rx_config.h」ファイル内の設定が誤っている可能性があります。
「r_cmtw_rx_config.h」ファイルを確認してください。誤った設定が存在している場合、その設定にとって正しい値を設定してください。詳細については、「2.9 コンパイル時の設定」を参照してください。

(4) Q：期待通りの波形が出力されません。

A：ピンの設定が正しく実行されていない可能性があります。この FIT モジュールを使用するには、ピン設定を実行する必要があります。詳細については、「4 端子の設定」を参照してください。

7. 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
1.20	2016.10.1	—	初版発行
1.30	2017.07.21	--	FIT モジュールの RX65N グループ（ROM 2MB 版）対応
		5	「2.5 対応ツールチェーン」に RXC v2.06.00、RXC v2.07.00 を追加
		6	「2.6 割り込みベクタ」を追加
		12	「2.13 FIT モジュールをプロジェクトに追加する方法」を更新
		22	4. 端子設定を更新
1.31	2017.10.31	24	「5.3 cmtw_demo_rskrx65n」を追加
		24	「5.4 cmtw_demo_rskrx65n_2m」を追加
		25	「5.6 デモのダウンロード方法」を追加
		26	「6. 付録」を追加
1.32	2018.11.16	—	XML 内にドキュメント番号を追加。
		5	「2.5 対応ツールチェーン」に RXC v3.01.00 を追加
		12	「2.13 FIT モジュールをプロジェクトに追加する方法」を更新
		13	「2.14 for 文、while 文、do while 文について」を追加
		27	Rev.1.32 に対応する表を追加。
1.40	2019.02.01	7	割り込み優先レベルのマクロ名を訂正。
		15-22	各 API 関数で「Reentrant」の説明を削除。
2.00	2019.05.20	—	以下のコンパイラをサポート。 - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		1	「ターゲットコンパイラ」のセクションを追加。 関連ドキュメントを削除。
		5	「2.3 ソフトウェアの要求」r_bsp v5.20 以上が必要
		8	「2.10 コードサイズ」セクションを更新。
		28	表 6.1「動作確認環境」： Rev.2.00 に対応する表を追加。
		31	「Web サイトおよびサポート」のセクションを削除。
		プログラム	GCC と IAR コンパイラに関して、以下を変更。 1. R_CMTW_GetVersion 関数のインライン展開を削除。 2. 「evenaccess」を、BSP のマクロ定義で置き換えた。 3. 割り込み関数の宣言を、BSP のマクロ定義で置き換えた。
2.10	2019.08.15	1、6	RX72M のサポートを追加。
		9	RX72M に対応するコードサイズを追加。
		28	「6.1 動作確認環境」： Rev.2.10 に対応する表を追加。
		プログラム	RX72M のサポートを追加。

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。