RX ファミリ

RSCAN 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 RSCAN モジュールについて説明します。メールボックス（保持可能なメッセージ数は 1）、FIFO（保持可能なメッセージ数は 4）、またはこれらを組み合わせてメッセージを転送できます。

以降、本モジュールを RSCAN FIT モジュールと称します。

注: **E1 エミュレータを使用してアプリケーションを開発する場合で、E1 エミュレータからボードに電源を供給する場合は、デバッグ構成で示される 3.3V ではなく 5.0V を供給してください。5.0V が供給されない場合は RSCAN が正しく動作しません。**

対象デバイス

- RX140 グループ (ROM 容量が 128K バイト以上の製品)
- RX230 グループ、RX231 グループ
- RX23E-A グループ
- RX23W グループ
- RX24T グループ
- RX24U 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 FIT モジュールは、RZ/A1 で提供される RSCAN ドライバと互換性があります。RX100、RX200 シリーズの CAN モジュールは 1 チャネルですが、本 API も同様です。メールボックスと FIFO（ボックス）の静的配置は、RZ/A1 と同様の配置にハードコーディングされていますが、使用できるリソースは限られています。

メールボックスで保持可能なメッセージ数は 1 で、送信、受信ともに 4 つのメールボックスがあります。送信メールボックスは任意で割り込み動作を設定できますが、受信メールボックスはできません。送信メールボックスは、前のメッセージが送信されるまで次のメッセージの送信を受け付けません。受信メールボックスでは、エラーが生成されることなく前のメッセージが上書きされますので、ボックスには常に最新の受信メッセージが入っています。ハードウェア割り込みオプションは使用できません。

送受信 FIFO で保持可能なメッセージ数は 4 です。FIFO は、メールボックスと同様に、メッセージの送受信に使用されます。送受信 FIFO は任意で割り込み駆動に設定できます。受信 FIFO をメッセージ受信ごとの割り込みに設定すると、割り込みをサポートしている受信メールボックスと同様の動きになります。
送信履歴 FIFO という特殊な FIFO があり、保持可能なメッセージ数は 8 です。履歴 FIFO には R_CAN_SendMsg()でタグ付けされたすべてのメッセージが送信順に記録されます。
いずれの FIFO の使用も任意で、通常の動作に FIFO は必要ありません。

RSCAN ハードウェアは、バスに送信されるすべてのメッセージを処理します。このとき、どのメッセージを保持し、どのメッセージを無視するのかは受信規則を使って判断します。初めにメッセージをフィルタにかけて、メッセージが保持対象かどうかを確認します。次にメッセージ転送に使用するボックス（受信メールボックス、または受信 FIFO）を指定します。ハードウェアがメッセージをボックスに転送すると、R_CAN_GetMsg()関数を使って、ボックスからメッセージを読み出します。

割り込みはグローバル割り込みとチャネル割り込みの 2 種類を使用できます。受信 FIFO がメッセージを受信した場合、または、グローバルエラーが発生した場合はグローバル割り込みで示されます。グローバル割り込みは r_rscan_rx_config.h で有効にします。本 FIT モジュールが割り込みを検出し、R_CAN_Open()で指定されたユーザコールバック関数を呼び出して、発生したイベントを処理します。チャネル割り込みは、複数の送信条件、およびチャネルエラーに対応します。チャネル割り込みも、r_rscan_rx_config.h で有効にします。本 FIT モジュールが割り込みを検出し、R_CAN_InitChan()で指定されたユーザコールバック関数を呼び出して、発生したイベントを処理します。

以下の場合に使用される割り込みが、デフォルトで許可に設定されています。

- 受信、送信、履歴 FIFO のしきい値に達した
- 受信、送信、履歴 FIFO でオーバフローが発生した
- チャネルがエラーパッシブ状態に遷移した
- チャネルがバスオフ状態に遷移した
- チャネルがバスオフ状態から復帰した

以下の関数呼び出しシーケンスを使用して CAN を設定します。

```
R_CAN_Open();
R_CAN_InitChan();     // 1 チャネルに対して呼び出し
R_CAN_ConfigFIFO();   // 0 以上の FIFO に対して呼び出し
R_CAN_AddRxRule();    // 1～16 規則に対して呼び出し
```

CAN の設定後、CAN モジュールは通常の通信モードかテストモードに遷移します。

```
R_CAN_Control();      // CAN_CMD_SET_MODE_COMM または CAN_CMD_SET_MODE_TST_xxx を使用
```

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。

- RSCAN

2.2 ハードウェアリソースの要求

RSCAN に加えて、以下が要求されます。

- CAN チャネルに割り当てられた端子 2 本

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）v5.20 以上

2.4 制限事項

RSCAN のすべての機能は使用されません。以下に使用されない機能を示します。

- 送受信 FIFO の保持可能なメッセージ数の設定（1～16 まで設定可能だが、送受信とも 4 に固定）
- ID 優先送信（メールボックス番号によって送信。0 が最優先番号）
- 送信 FIFO インターバル送信
- ミラー機能
- ミラー機能該当メッセージの抽出
- DLC 置き換え機能
- 受信メッセージの転送先選択（転送先は 3 つ選択できるが、1 つに固定）
- バスオフ状態からの復帰方法選択（ISO11898-1 規格準拠）
- バスオフ強制復帰機能
- プロトコルエラーフラグ（蓄積／最初の発生）の選択：
  （全チャネルに対して蓄積でハードコーディング）

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 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_rscan_rx_if.h に記載されています。

r_rscan_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。

2.7 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。

2.8 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_rscan_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。

コンフィギュレーションオプション（r_rscan_rx_config.h）(1/3)

定義	ﾃﾞﾌｫﾙﾄ値	説明
CAN_CFG_PARAM_CHECKING_ENABLE	1	この定義を“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。
CAN_CFG_CLOCK_SOURCE	0	この定義を“0”に設定すると､CAN のクロックソースは周辺クロック(clkc)の 2 分周になります。“1”に設定すると、クロックソースは CAN の外部クロック (clk_xincan) になります。
CAN_CFG_INT_PRIORITY	5	すべての CAN 割り込み(0～31)の優先レベル
CAN_CFG_INT_RXFIFO_THRESHOLD	1	この定義を“0”に設定すると、受信 FIFO のしきい値に達したときの割り込みを無効にします。 “1”に設定すると、割り込みが有効になります。 R_CAN_ConfigFIFO()で FIFO の初期化を要求します。 割り込み処理でメインコールバック関数に“CAN_EVT_RXFIFO_THRESHOLD”を渡します。
CAN_CFG_INT_DLC_ERR	0	この定義を“0”に設定すると､DLC エラーを検出したときの割り込みを無効にします。 “1”に設定すると、割り込みが有効になります。 割り込み処理でメインコールバック関数に“CAN_EVT_GLOBAL_ERR”を渡します。
CAN_CFG_INT_FIFO_OVFL	1	この定義を“0”に設定すると、送信、ゲートウェイ、または受信 FIFO がオーバフローしたときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 R_CAN_ConfigFIFO()で FIFO の初期化を要求します。 割り込み処理でメインコールバック関数に“CAN_EVT_GLOBAL_ERR”を渡します。

コンフィギュレーションオプション（r_rscan_rx_config.h）(2/3)		
定義	ﾃﾞﾌｫﾙﾄ値	説明
CAN_CFG_INT_HIST_FIFO_OVFL	1	この定義を“0”に設定すると、履歴 FIFO がオーバフローしたときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 R_CAN_ConfigFIFO()で FIFO の初期化を要求します。 割り込み処理でメインコールバック関数に“CAN_EVT_GLOBAL_ERR”を渡します。
CAN_CFG_INT_TXFIFO_THRESHOLD	1	この定義を“0”に設定すると、送信 FIFO のしきい値に達したときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 R_CAN_ConfigFIFO()で FIFO の初期化を要求します。 割り込み処理でチャネルコールバック関数に“CAN_EVT_TRANSMIT”を渡します。
CAN_CFG_INT_HIST_FIFO_THRESHOLD	1	この定義を“0”に設定すると、履歴 FIFO のしきい値に達したときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 R_CAN_ConfigFIFO()で FIFO の初期化を要求します。 割り込み処理でチャネルコールバック関数に“CAN_EVT_TRANSMIT”を渡します。
CAN_CFG_INT_MBX_TX_COMPLETE	0	この定義を“0”に設定すると、メールボックスが送信を完了したときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 割り込み処理でチャネルコールバック関数に“CAN_EVT_TRANSMIT”を渡します。
CAN_CFG_INT_MBX_TX_ABORTED	0	この定義を“0”に設定すると、メールボックスの送信が中止されたときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 割り込み処理でチャネルコールバック関数に“CAN_EVT_TRANSMIT”を渡します。
CAN_CFG_INT_BUS_ERROR	0	この定義を“0”に設定すると、バスエラーを検出したときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 割り込み処理でチャネルコールバック関数に“CAN_EVT_CHANNEL_ERR”を渡します。
CAN_CFG_INT_ERR_WARNING	0	この定義を“0”に設定すると、エラーワーニングを検出したときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 割り込み処理でチャネルコールバック関数に“CAN_EVT_CHANNEL_ERR”を渡します。
CAN_CFG_INT_ERR_PASSIVE	1	この定義を“0”に設定すると、エラーパッシブを検出したときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 割り込み処理でチャネルコールバック関数に“CAN_EVT_CHANNEL_ERR”を渡します。
CAN_CFG_INT_BUS_OFF_ENTRY	1	この定義を“0”に設定すると、バスオフエラーを検出したときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 割り込み処理でチャネルコールバック関数に“CAN_EVT_CHANNEL_ERR”を渡します。

コンフィギュレーションオプション（r_rscan_rx_config.h）(3/3)		
定義	ﾃﾞﾌｫﾙﾄ値	説明
CAN_CFG_INT_BUS_OFF_RECOVERY	1	この定義を“0”に設定すると、バスオフ復帰を検出したときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 割り込み処理でチャネルコールバック関数に“CAN_EVT_CHANNEL_ERR”を渡します。
CAN_CFG_INT_OVERLOAD_FRAME_TX	0	この定義を“0”に設定すると、オーバーロードを検出したときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 割り込み処理でチャネルコールバック関数に“CAN_EVT_CHANNEL_ERR”を渡します。
CAN_CFG_INT_BUS_LOCK	0	この定義を“0”に設定すると、バスロックを検出したときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 割り込み処理でチャネルコールバック関数に“CAN_EVT_CHANNEL_ERR”を渡します。
CAN_CFG_INT_ARB_LOST	0	この定義を“0”に設定すると、アービトレーションロストを検出したときの割り込みを無効にします。 “1”を設定すると、割り込みが有効になります。 割り込み処理でチャネルコールバック関数に“CAN_EVT_CHANNEL_ERR”を渡します。

2.9 コードサイズ

ツールチェーン（セクション 2.5 に記載）でのコードサイズは、最適化レベル 2 です。コードサイズには、すべての割り込み処理（アクティブ、または非アクティブに設定）、および FIFO 対応の全コードが含まれます。

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「2.8 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。掲載した値は、「2.5 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。

ROM、RAM およびスタックのコードサイズ

デバイス	カテゴリ	使用メモリ					
		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処理あり	パラメータチェック処理なし	パラメータチェック処理あり	パラメータチェック処理なし	パラメータチェック処理あり	パラメータチェック処理なし
RX231	ROM	3211 バイト	2683 バイト	5824 バイト	4976 バイト	5547 バイト	4916 バイト
	RAM	20 バイト	20 バイト	20 バイト	20 バイト	20 バイト	20 バイト
	スタック	36 バイト	36 バイト	-	-	144 バイト	144 バイト
RX23W	ROM	3211 バイト	2683 バイト	-	-	-	-
	RAM	20 バイト	20 バイト	-	-	-	-
	スタック	72 バイト	72 バイト	-	-	-	-
RX23E-A	ROM	3237 バイト	2723 バイト	6104 バイト	5232 バイト	5421 バイト	4782 バイト
	RAM	20 バイト	20 バイト	20 バイト	20 バイト	24 バイト	24 バイト
	スタック	100 バイト	100 バイト	-	-	96 バイト	96 バイト
RX140 (ROM 容量が 128K バイト以上の製品)	ROM	3204 バイト	2710 バイト	6088 バイト	5208 バイト	5296 バイト	4661 バイト
	RAM	20 バイト	20 バイト	0 バイト	0 バイト	20 バイト	20 バイト
	スタック	68 バイト	68 バイト	-	-	124 バイト	124 バイト

2.10 API データ構造体

API で使用されるデータ構造体を示します。

2.10.1 ボックス ID（メールボックスと FIFO）

```
typedef enum e_can_box
{
    CAN_BOX_NONE            = 0,  //未使用のパラメータ値

    CAN_BOX_CH0_TXMBX_0      = (CAN_FLG_TXMBX | 0),
    CAN_BOX_CH0_TXMBX_1      = (CAN_FLG_TXMBX | 1),
    CAN_BOX_CH0_TXMBX_2      = (CAN_FLG_TXMBX | 2),
    CAN_BOX_CH0_TXMBX_3      = (CAN_FLG_TXMBX | 3),

    CAN_BOX_RXMBX_0         = (CAN_FLG_RXMBX | 0),
    CAN_BOX_RXMBX_1         = (CAN_FLG_RXMBX | 1),
    CAN_BOX_RXMBX_2         = (CAN_FLG_RXMBX | 2),
    CAN_BOX_RXMBX_3         = (CAN_FLG_RXMBX | 3),

    CAN_BOX_RXFIFO_0        = (CAN_FLG_FIFO | CAN_MASK_RXFIFO_0),
    CAN_BOX_RXFIFO_1        = (CAN_FLG_FIFO | CAN_MASK_RXFIFO_1),

    CAN_BOX_TXFIFO          = (CAN_FLG_FIFO | CAN_MASK_CH0_TXFIFO_0),

    CAN_BOX_HIST_FIFO       = (CAN_FLG_FIFO | CAN_MASK_CH0_HIST_FIFO),
} can_box_t;
```

2.10.2 R_CAN_Open()データ型

```
typedef enum e_can_timestamp_src
{
    CAN_TIMESTAMP_SRC_HALF_PCLK = 0,
    CAN_TIMESTAMP_SRC_CANMCLK   = 1,    // EXTAL 端子から取得
    CAN_TIMESTAMP_SRC_END_ENUM
} can_timestamp_src_t;

typedef enum e_can_timestamp_div
{
    CAN_TIMESTAMP_DIV_1    = 0,
    CAN_TIMESTAMP_DIV_2    = 1,
    CAN_TIMESTAMP_DIV_4    = 2,
    CAN_TIMESTAMP_DIV_8    = 3,
    CAN_TIMESTAMP_DIV_16   = 4,
    CAN_TIMESTAMP_DIV_32   = 5,
    CAN_TIMESTAMP_DIV_64   = 6,
    CAN_TIMESTAMP_DIV_128  = 7,
    CAN_TIMESTAMP_DIV_256  = 8,
    CAN_TIMESTAMP_DIV_512  = 9,
    CAN_TIMESTAMP_DIV_1024 = 10,
    CAN_TIMESTAMP_DIV_2048 = 11,
    CAN_TIMESTAMP_DIV_4096 = 12,
    CAN_TIMESTAMP_DIV_8192 = 13,
    CAN_TIMESTAMP_DIV_16384 = 14,
    CAN_TIMESTAMP_DIV_32768 = 15,
    CAN_TIMESTAMP_DIV_END_ENUM
} can_timestamp_div_t;

typedef struct st_can_cfg
{
    can_timestamp_src_t    timestamp_src;
    can_timestamp_div_t    timestamp_div;
} can_cfg_t;
```

2.10.3 コールバック関数イベント

```
typedef enum e_can_cb_evt     // コールバック関数イベント
{
    // メインコールバックイベント
    CAN_EVT_RXFIFO_THRESHOLD,  // 受信 FIFO しきい値
    CAN_EVT_GLOBAL_ERR,        // 受信 FIFO または履歴 FIFO でオーバフロー／DLC エラー

    // チャネルコールバックイベント
    CAN_EVT_TRANSMIT,           // メールボックス送信完了、または送信中止,
                                // 送信 FIFO または履歴 FIFO のしきい値
    CAN_EVT_CHANNEL_ERR,
} can_cb_evt_t;
```

2.10.4 R_CAN_InitChan()データ型

```
typedef struct st_can_bitrate
{
    uint16_t    prescaler;  // 1-1024
    uint8_t     tseg1;      // 4-16
    uint8_t     tseg2;      // 2-8
    uint8_t     sjw;        // 1-4
} can_bitrate_t;

/* 500kbps でのサンプル設定*/
#define CAN_RSK_27MHZ_PCLKB_500KBPS_PRESCALER   3
#define CAN_RSK_27MHZ_PCLKB_500KBPS_TSEG1       5
#define CAN_RSK_27MHZ_PCLKB_500KBPS_TSEG2       3
#define CAN_RSK_27MHZ_PCLKB_500KBPS_SJW         1

#define CAN_RSK_32MHZ_PCLKB_500KBPS_PRESCALER    2
#define CAN_RSK_32MHZ_PCLKB_500KBPS_TSEG1       11
#define CAN_RSK_32MHZ_PCLKB_500KBPS_TSEG2        4
#define CAN_RSK_32MHZ_PCLKB_500KBPS_SJW          4
                                                        //他の設定
#define CAN_RSK_8MHZ_XTAL_500KBPS_PRESCALER      1              // 2
#define CAN_RSK_8MHZ_XTAL_500KBPS_TSEG1         10        // 5
#define CAN_RSK_8MHZ_XTAL_500KBPS_TSEG2          5        // 2
#define CAN_RSK_8MHZ_XTAL_500KBPS_SJW            1        // 1
```

2.10.5 R_CAN_ConfigFIFO()データ型

```
typedef enum e_can_fifo_threshold    // 注: 履歴 FIFO のしきい値は 1 または 6 のみ
{                                    //
    CAN_FIFO_THRESHOLD_2    = 3,     // 4/8 of 4
    CAN_FIFO_THRESHOLD_3    = 5,     // 6/8 of 4
    CAN_FIFO_THRESHOLD_6    = 5,     // 履歴 FIFO のみ!
    CAN_FIFO_THRESHOLD_FULL = 7,     // 8/8 of 4
    CAN_FIFO_THRESHOLD_1    = 8,     // すべてのメッセージ
    CAN_FIFO_THRESHOLD_END_ENUM
} can_fifo_threshold_t;
```

2.10.6 R_CAN_AddRxRule()データ型

```
typedef struct st_can_filter
{
    uint8_t     check_ide:1;
    uint8_t     ide;
    uint8_t     check_rtr:1;
    uint8_t     rtr;
    uint32_t    id;
    uint32_t    id_mask;
    uint8_t     min_dlc;
    uint16_t    label;         // 12 ビットラベル
} can_filter_t;
```

2.10.7 R_CAN_SendMsg()データ型

```
typedef struct st_can_txmsg
{
   uint8_t    ide;
   uint8_t    rtr;
   uint32_t   id;
   uint8_t    dlc;
   uint8_t    data[8];
   bool_t     one_shot;      // エラー時リトライなし; 送信メールボックスのみ
   bool_t     log_history;   // 履歴が必要な場合は “true”
   uint8_t    label;         // 履歴 FIFO では 8 ビットラベル
} can_txmsg_t;
```

2.10.8 R_CAN_GetMsg()データ型

```
typedef struct st_can_rxmsg
{
   uint8_t    ide;
   uint8_t    rtr;
   uint32_t   id;
   uint8_t    dlc;
   uint8_t    data[8];
   uint16_t   label;         //受信規則により 12 ビットラベル
   uint16_t   timestamp;
} can_rxmsg_t;
```

2.10.9 R_CAN_GetHistoryEntry()データ型

```
typedef struct st_can_history
{
   can_box_t   box_id;       // メッセージ送信に使用したボックス
   uint8_t     label;        // 送信履歴データ情報: 8 ビットラベル
} can_history_t;
```

2.10.10 R_CAN_GetStatusMask()データ型

```
typedef enum e_can_stat
{
   CAN_STAT_FIFO_EMPTY,
   CAN_STAT_FIFO_THRESHOLD,
   CAN_STAT_FIFO_OVFL,         // 読み出し後、ビットをリセット
   CAN_STAT_RXMBX_FULL,
   CAN_STAT_GLOBAL_ERR,        // 読み出し後、DLC エラービットをリセット
   CAN_STAT_CH_TXMBX_SENT,     // 読み出し後、ビットをリセット
   CAN_STAT_CH_TXMBX_ABORTED, // 読み出し後、ビットをリセット
   CAN_STAT_CH_ERROR,          // 読み出し後、ビットをリセット
   CAN_STAT_END_ENUM
} can_stat_t;

/* マスク値を返す (同時に複数ビットの設定が可能)

/* CAN_STAT_CH_TXMBX_SENT, CAN_STAT_CH_TXMBX_ABORTED */
#define CAN_MASK_TXMBX_0          (0x0001)
#define CAN_MASK_TXMBX_1          (0x0002)
#define CAN_MASK_TXMBX_2          (0x0004)
#define CAN_MASK_TXMBX_3          (0x0008)

/* CAN_STAT_RXMBX_FULL */
#define CAN_MASK_RXMBX_0          (0x0001)
#define CAN_MASK_RXMBX_1          (0x0002)
#define CAN_MASK_RXMBX_2          (0x0004)
#define CAN_MASK_RXMBX_3          (0x0008)
```

```
/* CAN_STAT_FIFO_EMPTY, CAN_STAT_FIFO_THRESHOLD, CAN_STAT_FIFO_OVFL */
#define CAN_MASK_RXFIFO_0          (0x00000001)
#define CAN_MASK_RXFIFO_1          (0x00000002)
#define CAN_MASK_TXFIFO             (0x00000100)
#define CAN_MASK_HIST_FIFO         (0x00800000)

/* CAN_STAT_GLOBAL_ERR */
#define CAN_MASK_ERR_DLC            (0x0001)
#define CAN_MASK_ERR_RX_OVFL        (0x0002)
#define CAN_MASK_ERR_HIST_OVFL      (0x0004)
#define CAN_MASK_ERR_FIFO_OVFL      (0x0006)

/* CAN_STAT_CH_ERROR */
#define CAN_MASK_ERR_PROTOCOL       (0x0001)
#define CAN_MASK_ERR_WARNING        (0x0002)
#define CAN_MASK_ERR_PASSIVE        (0x0004)
#define CAN_MASK_ERR_BUS_OFF_ENTRY (0x0008)
#define CAN_MASK_ERR_BUS_OFF_EXIT  (0x0010)
#define CAN_MASK_ERR_OVERLOAD       (0x0020)
#define CAN_MASK_ERR_DOMINANT_LOCK (0x0040)
#define CAN_MASK_ERR_ARB_LOST       (0x0080)
#define CAN_MASK_ERR_STUFF          (0x0100)
#define CAN_MASK_ERR_FORM           (0x0200)
#define CAN_MASK_ERR_ACK            (0x0400)
#define CAN_MASK_ERR_CRC            (0x0800)
#define CAN_MASK_ERR_RECESSIVE_BIT (0x1000)
#define CAN_MASK_ERR_DOMINANT_BIT  (0x2000)
#define CAN_MASK_ERR_ACK_DELIMITER (0x4000)
```

2.10.11 R_CAN_GetCountErr()データ型

```
typedef enum e_can_count
{
    CAN_COUNT_RX_ERR,
    CAN_COUNT_TX_ERR,
    CAN_COUNT_END_ENUM
} can_count_t;
```

2.10.12 R_CAN_Control()データ型

```
typedef enum e_can_cmd
{
    CAN_CMD_ABORT_TX,                   // 引数: 送信メールボックス ID
    CAN_CMD_RESET_TIMESTAMP,
    CAN_CMD_SET_MODE_COMM,              // ノーマルバス通信開始
    CAN_CMD_SET_MODE_TST_STANDARD,
    CAN_CMD_SET_MODE_TST_LISTEN,
    CAN_CMD_SET_MODE_TST_EXT_LOOPBACK,
    CAN_CMD_SET_MODE_TST_INT_LOOPBACK,
    CAN_CMD_END_ENUM
} can_cmd_t;
```

2.11 戻り値

API 関数の戻り値を示します。この列挙型は、API 関数の宣言と共に r_rscan_rx_if.h に記載されています。

```
typedef enum e_can_err          // CAN API エラーコード
{
CAN_SUCCESS=0,
CAN_ERR_OPENED,                 // Open 関数は既に呼び出されています。
CAN_ERR_NOT_OPENED,             // Open 関数は呼び出されていません。
CAN_ERR_INIT_DONE,              // InitChan 関数は既に呼び出されています。
CAN_ERR_CH_NO_INIT,             // チャネルは初期化されていません。
CAN_ERR_INVALID_ARG,            // 無効な引数が渡されました。
CAN_ERR_MISSING_CALLBACK,       // コールバック関数が提供されずに、割り込みが要求されました。
CAN_ERR_MAX_RULES,              // 規則が最大数に達しています。
CAN_ERR_BOX_FULL,               // 送信メールボックス、または FIFO がフルです。
CAN_ERR_BOX_EMPTY,              // 受診メールボックス、または FIFO がフルです。
CAN_ERR_ILLEGAL_MODE            // 要求に対して不適切なモードです。
} can_err_t;
```

2.12 FIT モジュールの追加方法

本モジュールは、e^2 studio で、使用するプロジェクトごとに追加する必要があります。

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

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

FIT プラグインを使用して FIT モジュールを追加する方法は、アプリケーションノート「e^2 studio に組み込む方法(R01AN1723)」の「3. FIT プラグインを使用して FIT モジュールをプロジェクトに追加する方法」を参照してください。

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

FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685)」を参照してください。

3. API 関数

概要

本 FIT モジュールには以下の関数が含まれます。

関数	説明
R_CAN_Open()	RSCAN FIT モジュールの内部構成とすべての受信メールボックスを初期化します。
R_CAN_InitChan()	チャネルのビットレートクロックを設定して、すべての送信メールボックスを初期化します。
R_CAN_ConfigFIFO()	使用する FIFO を初期化します。FIFO を使用しない場合は、本関数を呼び出す必要はありません。
R_CAN_AddRxRule()	チャネルへの受信規則を追加します。受信メッセージフィルタ、および転送先を指定します。
R_CAN_Control()	特殊な動作とモードの変更を扱う関数です。
R_CAN_SendMsg()	送信メールボックス、または送信 FIFO にメッセージを配置します。
R_CAN_GetMsg()	受信メールボックス、または受信 FIFO からメッセージを取得します。
R_CAN_GetHistoryEntry()	送信履歴 FIFO からログエントリを取得します。
R_CAN_GetStatusMask()	要求されたステータスに応じて 32 ビットマスクを返します。ビットの#define は“CAN_MASK_xxx”の形式を取ります。
R_CAN_GetCountFIFO()	FIFO 内のメッセージ数を返します。
R_CAN_GetCountErr()	送信エラー数、または受信エラー数を返します。
R_CAN_Close()	CAN モジュールを終了し、関連する割り込みを無効にします。
R_CAN_GetVersion()	本 FIT モジュールのバージョン番号を返します。

R_CAN_Open()

RSCAN FIT モジュールの内部構成とすべての受信メールボックスを初期化します。

Format

```
can_err_t R_CAN_Open(can_cfg_t *p_cfg,
                  void (* const p_callback)(can_cb_evt_t event,
                                            void         *p_args));
```

Parameters

p_cfg

設定用構造体へのポインタ。構成要素の型については 2.10.1 を参照してください。

```
typedef struct st_can_cfg
{
  can_timestamp_src_t ts_source;
  can_timestamp_div_t ts_divisor;
} can_cfg_t;
```

p_callback

メインコールバック関数への任意のポインタ。r_rscan_rx_config.h で受信 FIFO またはグローバルエラーに使用する割り込みが有効な場合は、本引数を設定してください。

event

コールバック関数の第一引数。割り込み要因を設定（2.10.3 参照）

p_args

コールバック関数の第二引数。（未使用）

Return Values

CAN_SUCCESS	*/*正常動作 */*
CAN_ERR_OPENED	*/* Open 関数は既に呼び出されています。*/*
CAN_ERR_INVALID_ARG	*/* p_cfg 構造体のメンバに無効な値が含まれます。*/*
CAN_ERR_MISSING_CALLBACK	*/*コールバック関数が提供されずに、config.h で*/*
	*/*メインコールバック割り込みが有効になっています。*/*

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

本関数は RSCAN FIT モジュールの内部構成を初期化し、CAN モジュールにクロックを提供して、グローバルモードとチャネルモードを「リセット」に設定します。引数“p_cfg”に応じてタイムスタンプを設定し、すべての受信メールボックスを初期化します。

r_rscan_rx_config.h で受信 FIFO のしきい値、DLC、または FIFO オーバフローエラーに使用する割り込みが有効な場合、コールバック関数を提供する必要があります。これらの割り込みが無効な場合は NULL を入力します。

Reentrant

この関数は再入不可です。

Example: ポーリングの設定

```
    /* r_rscan_rx_config.h でメインコールバックの割り込み要因をすべて“0”に設定  */
    can_cfg_t    config;
    can_err_t    err;

    /* タイムスタンプおよび Open 関数関連の設定 */
    config.timestamp_src = CAN_TIMESTAMP_SRC_HALF_PCLK;
    config.timestamp_div = CAN_TIMESTAMP_DIV_1024;
    err = R_CAN_Open(&config, NULL);
```

Example: 割り込みの設定

```
    /* r_rscan_rx_config.h でメインコールバックの割り込み要因を“1”に設定 */

    can_cfg_t    config;
    can_err_t    err;

    /* タイムスタンプおよび Open 関数関連の設定*/
    config.timestamp_src = CAN_TIMESTAMP_SRC_HALF_PCLK;
    config.timestamp_div = CAN_TIMESTAMP_DIV_1024;
    err = R_CAN_Open(&config, MyCallback);
```

```
/* サンプルコールバック関数 */
void MyCallback(can_cb_evt_t event, void *p_args)
{
uint32_t   mask;
can_err_t  err;

    if (event == CAN_EVT_RXFIFO_THRESHOLD)
    {
        mask = R_CAN_GetStatusMask(CAN_STAT_FIFO_THRESHOLD, NULL, &err);

        /* 使用中の RXFIFO を確認 */
        if (mask & CAN_MASK_RXFIFO_1)
        {
            /* メッセージ読み出し */
        }
    }
    else if (event == CAN_EVT_GLOBAL_ERR)
    {
        mask = R_CAN_GetStatusMask(CAN_STAT_GLOBAL_ERR, NULL, &err);

        if (mask & CAN_MASK_ERR_DLC)
        {
            /* DLC エラーの処理 */
        }

        if (mask & CAN_MASK_ERR_FIFO_OVFL)
        {
            mask = R_CAN_GetStatusMask(CAN_STAT_FIFO_OVFL, NULL, &err);

            /* 使用中の RXFIFO, GWFIFO, HIST_FIFO を確認 */
              if (mask & CAN_MASK_HIST_FIFO)
            {
                /* エラー処理 */
            }
        }
    }
}
```

Special Notes:
RSCAN で使用されるポート端子は、R_CAN_Open()を呼び出す前に初期化しておく必要があります。
以下に例を示します。

RX231:

```
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);

PORT5.PODR.BIT.B5 = 1;
PORT5.PODR.BIT.B4 = 0;
MPC.P54PFS.BYTE = 0x10; // 端子機能選択 P54 CTXD0
MPC.P55PFS.BYTE = 0x10; // 端子機能選択 P55 CRXD0
PORT5.PDR.BIT.B4 = 1;   // TX 端子の方向を出力に設定
PORT5.DSCR.BIT.B4 = 1;  // High 出力
PORT5.PDR.BIT.B5 = 0;   // RX 端子の方向を入力に設定(デフォルト)
PORT5.PMR.BIT.B4 = 1;   // TX 端子のモードを周辺機能に設定
PORT5.PMR.BIT.B5 = 1;   // RX 端子のモードを周辺機能に設定

R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_MPC);
```

RX24T:

```
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);

PORTA.PODR.BIT.B1 = 1;
PORTA.PODR.BIT.B0 = 0;
MPC.PA0PFS.BYTE = 0x10; // 端子機能選択 PA0 CTXD0
MPC.PA1PFS.BYTE = 0x10; // 端子機能選択 PA1 CRXD0
PORTA.PDR.BIT.B0 = 1;   // TX 端子の方向を出力に設定
PORTA.DSCR.BIT.B0 = 1;  // High 出力
PORTA.PDR.BIT.B1 = 0;   // RX 端子の方向を入力に設定(デフォルト)
PORTA.PMR.BIT.B0 = 1;   // TX 端子のモードを周辺機能に設定
PORTA.PMR.BIT.B1 = 1;   // RX 端子のモードを周辺機能に設定

R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_MPC);
```

RX24U:

```
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);

PORTF.PODR.BIT.B3 = 1;
PORTF.PODR.BIT.B2 = 0;
MPC.PF2PFS.BYTE = 0x10; // 端子機能選択 PF2 CTXD0
MPC.PF3PFS.BYTE = 0x10; // 端子機能選択 PF3 CRXD0
PORTF.PDR.BIT.B2 = 1;   // TX 端子の方向を出力に設定
PORTF.DSCR.BIT.B2 = 1;  // High 出力
PORTF.PDR.BIT.B3 = 0;   // RX 端子の方向を入力に設定(デフォルト)
PORTF.PMR.BIT.B2 = 1;   // TX 端子のモードを周辺機能に設定
PORTF.PMR.BIT.B3 = 1;   // RX 端子のモードを周辺機能に設定

R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_MPC);
```

R_CAN_InitChan()

チャネルのビットレートクロックを設定し、すべての送信メールボックスを初期化します。

Format

```
can_err_t  R_CAN_InitChan(uint8_t       chan,
                          can_bitrate_t  *p_baud,
                          void   (* const p_chcallback)(uint8_t       chan,
                                                        can_cb_evt_t  event,
                                                        void          *p_args));
```

Parameters

chan

初期化するチャネル（“0”のみが有効）

p_baud

ビットレートの構造体へのポインタ。設定値の算出については、ユーザーズマニュアル ハードウェア編にて CAN モジュール章の「ビットタイミングの設定」を参照してください。r_rscan_rx_if.h でデフォルト値が提供されている設定もあります。

```
typedef struct st_can_bitrate
{
  uint16_t prescaler;
  uint8_t tseg1;
  uint8_t tseg2;
  uint8_t sjw;
} can_bitrate_t;
```

p_chcallback

チャネルのコールバック関数への任意ポインタ。r_rscan_rx_config.h で送信メールボックス、送信 FIFO、履歴 FIFO、またはバスエラーで使用する割り込みが有効な場合、この引数が必要です。

channel

チャネルコールバック関数の第一引数。割り込みが発生したチャネルを設定（常に“0”）

event

チャネルコールバック関数の第二引数。割り込み要因を設定（2.10.3 参照）

p_args

コールバック関数の第三引数（未使用）。

Return Values

CAN_SUCCESS	*/*正常動作 */*
CAN_ERR_ILLEGAL_MODE	*/*グローバルリセットモードではありません。*/*
	/（Open()関数が呼び出されていない可能性があります）。*/*
CAN_ERR_INVALID_ARG	*/*提供された引数は無効です。*/*
CAN_ERR_MISSING_CALLBACK	*/*コールバック関数が提供されていませんが、config.h で */*
	*/*チャネル割り込みが有効です。*/*

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

本関数はすべてのチャネルの送信メールボックスを初期化します。r_rscan_rx_config.h の設定に従って、ビットレートを設定し、割り込み要因を有効にします。p_baud のデフォルト値は r_rscan_rx_if.h で設定されます。

r_rscan_rx_config.h で送信メールボックス、送信 FIFO、履歴 FIFO、またはバスエラーで使用する割り込みが有効な場合、コールバック関数を提供する必要があります。コールバック関数が必要ない場合は“NULL”を設定します。

Reentrant

この関数は、チャネルが異なる場合、再入可能（リエントラント）です。

Example: ポーリングの設定

```
    /* r_rscan_rx_config.h でチャネル割り込み要因をすべて“0”に設定*/

    can_bitrate_t    baud;
    can_err_t       err;

    /* チャネル 0 の初期設定 (RSKRX231) */
    baud.prescaler = CAN_RSK_8MHZ_XTAL_500KBPS_PRESCALER;
    baud.tseg1 = CAN_RSK_8MHZ_XTAL_500KBPS_TSEG1;
    baud.tseg2 = CAN_RSK_8MHZ_XTAL_500KBPS_TSEG2;
    baud.sjw = CAN_RSK_8MHZ_XTAL_500KBPS_SJW;

    err = R_CAN_InitChan(CAN_CH0, &baud, NULL);
```

Example: 割り込みの設定

```
    /* r_rscan_rx_config.h でチャネル割り込み要因を“1”に設定 */

    can_bitrate_t    baud;
    can_err_t       err;

    /* チャネル 0 の初期設定 (RSKRX231) */
    baud.prescaler = CAN_RSK_8MHZ_XTAL_500KBPS_PRESCALER;
    baud.tseg1 = CAN_RSK_8MHZ_XTAL_500KPS_TSEG1;
    baud.tseg2 = CAN_RSK_8MHZ_XTAL_500KPS_TSEG2;
    baud.sjw = CAN_RSK_8MHZ_XTAL_500KPS_SJW;

    err = R_CAN_InitChan(CAN_CH0, &baud, MyChanCallback);
```

```
/* サンプルコールバック関数のテンプレート */
void MyChanCallback(uint8_t      chan,
                    can_cb_evt_t  event,
                    void         *p_args)
{
uint32_t   mask;
can_err_t  err;

    if (event == CAN_EVT_TRANSMIT)
    {
        mask = R_CAN_GetStatusMask(CAN_STAT_CH_TXMBX_SENT, chan, &err);

        /* 使用中の送信メールボックスを確認 */
        if (mask & CAN_MASK_TXMBX_3)
        {
            /* 処理を記載 */
        }

        mask = R_CAN_GetStatusMask(CAN_STAT_CH_TXMBX_ABORTED, chan, &err);

        /* 使用中の送信メールボックスを確認 */
```

```
        if (mask & CAN_MASK_TXMBX_0)
        {
            /* 処理を記載 */
        }

        mask = R_CAN_GetStatusMask(CAN_STAT_FIFO_THRESHOLD, NULL, &err);

        /* 使用中の送信 FIFO および履歴 FIFO を確認 */
        if (mask & CAN_MASK_TXFIFO)
        {
            /* 次の送信メッセージを読み出し */
        }
    }

    else if (event == CAN_EVT_CHANNEL_ERR)
    {
        mask = R_CAN_GetStatusMask(CAN_STAT_CH_ERROR, chan, &err);

        /* 必要に応じて個々のエラーを確認 */
        if (mask & CAN_MASK_ERR_BUS_OFF_ENTRY)
        {
            /* エラー処理 */
        }

        if (mask & CAN_MASK_ERR_BUS_OFF_EXIT)
        {
            /* 復帰処理 */
        }
    }
}
```

Special Notes:

なし

R_CAN_ConfigFIFO()

使用する FIFO を初期化します。FIFO を使用しない場合はこの関数を呼び出す必要はありません。

Format

```
can_err_t  R_CAN_ConfigFIFO(can_box_t               fifo,
                            can_fifo_threshold_t    threshold,
                            can_box_t               txmbx_id);
```

Parameters

fifo_id

FIFO のボックス ID（2.10.1 参照）

threshold

割り込みフラグを設定するために FIFO で必要なメッセージ数（2.10.5 参照）。履歴 FIFO に有効なエントリのしきい値は 1、または 6 のみです。その他の FIFO には 1、2、3、またはフル (4)を指定できます。

txmbx_id

関連付けする送信メールボックスのボックス ID（送信 FIFO にのみ適用）。受信および履歴 FIFO の場合、この引数は無視されます。

Return Values

CAN_SUCCESS	*/*正常動作 */*
CAN_ERR_ILLEGAL_MODE	*/*グローバルリセットモードではありません */*
	/（Open()関数が呼び出されていない可能性があります）。*/*
CAN_ERR_CH_NO_INIT	*/*チャネルが初期化されていません。*/*
CAN_ERR_INVALID_ARG	*/*無効な引数が提供されました。*/*

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

FIFO の使用は任意です。

本関数は FIFO の稼働に使用します。送受信 FIFO で保持できるメッセージ数は 4 です（履歴 FIFO は 8）。送信 FIFO は標準の送信メールボックスとの関連付けが必要です。メールボックス番号で、送信時の FIFO の優先順位が決まります（優先順位：メールボックス 0＝最上位、メールボックス 3＝最下位）。

Reentrant

この関数は、FIFO が異なる場合、再入可能（リエントラント）です。

Example: 受信 FIFO

```
    can_err_t       err;

    /*
     * 受信 FIFO 0 で受信したメッセージごとに割り込みフラグを設定。
     * config.h で CAN_CFG_INT_RXFIFO_THRESHOLD を“1”に設定すると、割り込み発生。
     * CAN_EVT_RXFIFO_THRESHOLD を使って、割り込みでメインのコールバック関数を呼び出す。
     */
    err = R_CAN_ConfigFIFO(CAN_BOX_RXFIFO_0,
                        CAN_FIFO_THRESHOLD_1,
                        CAN_BOX_NONE);              //未使用のフィールド
```

Example: 送信 FIFO

```
    can_err_t       err;

    /*
     * メールボックス 3 をチャネル 0 の送信 FIFO 0 と関連付ける。
     * FIFO の残メッセージが 2 になったら割り込みフラグを設定。
     * config.h で CAN_CFG_INT_TXFIFO_THRESHOLD を “1”に設定すると、割り込み発生。
     * CAN_EVT_TRANSMIT を使って、割り込みでチャネルのコールバック関数を呼び出す。
     */
    err = R_CAN_ConfigFIFO(CAN_BOX_TXFIFO,
                           CAN_FIFO_THRESHOLD_2,
                           CAN_BOX_CH0_TXMBX_3);
```

Example: 履歴 FIFO

```
  can_err_t       err;

 /*
  * 履歴 FIFO のしきい値を 6 に設定
  * config.h で CAN_CFG_INT_HIST_FIFO_THRESHOLD を “1”に設定すると、割り込み発生。
  * CAN_EVT_TRANSMIT を使って、割り込みでチャネルのコールバック関数を呼び出す。
  */
  err = R_CAN_ConfigFIFO(CAN_BOX_HIST_FIFO,
                      CAN_FIFO_THRESHOLD_6,
                      CAN_BOX_NONE);                // 未使用フィールド
```

Special Notes:

なし

R_CAN_AddRxRule()

チャネルへの受信規則を追加します。受信メッセージフィルタ、および転送先を指定します。

Format

```
can_err_t  R_CAN_AddRxRule(uint8_t           chan,
                           can_filter_t      *p_filter,
                           can_box_t         dst_box);
```

Parameters

chan

規則を適用するチャネル（常に“0”）

p_filter

規則情報へのポインタ

```
typedef struct st_can_filter
{
  uint8_t      check_ide:1;
  uint8_t      ide;
  uint8_t      check_rtr:1;
  uint8_t      rtr;
  uint32_t     id;
  uint32_t     id_mask;
  uint8_t      min_dlc;
  uint16_t     label;        // 12 ビットラベル
} can_filter_t;
```

dst_box

メッセージの転送先ボックス（受信メールボックスまたは受信 FIFO）（2.10.1 参照）

Return Values

CAN_SUCCESS	*/*正常動作 */*
CAN_ERR_ILLEGAL_MODE	*/*グローバルリセットモードではありません*/*
	/（Open()関数が呼び出されていない可能性があります）。*/*
CAN_ERR_CH_NO_INIT	*/*チャネルは初期化されていません。*/*
CAN_ERR_INVALID_ARG	*/*無効な引数が提供されました。*/*
CAN_ERR_MAX_RULES	*/*規則が最大数に達しています（r_rscan_rx_config.h で定義）*/*

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

本関数はチャネルに受信規則を追加するのに使用されます。最初に、受信メッセージのどのフィールドを検査するかによって、フィルタを設定します。フィルタテストに合格したら、メッセージの転送先を設定します。

id_mask フィールドが“1”の場合、受信メッセージ ID の対応するビットが、このフィルタの id フィールドのビットに対してチェックされます（以下の Examples 参照）。

規則の label フィールドは任意のフィールドで、フィルタを渡す各メッセージと関連しています。R_CAN_GetMsg()を使って受信ボックス（メールボックス、または FIFO）からメッセージを取得する時に、label フィールドはメッセージの簡易的な識別子とし提供されます。

Reentrant

この関数は再入不可です。

Example 1: メッセージ範囲の一致

```
can_filter_t  filter;
can_err_t     err;

/* フィルタの設定 */
filter.check_ide = 1;        // メッセージの IDE フィールドをチェック
filter.ide = 0;                    // 11 ビット ID
filter.check_rtr = 0;        // メッセージの RTR フィールドはチェックしない
filter.rtr = 0;                    // (チェックしないので、値は何でも良い)
filter.id = 0x040;           // メッセージ ID
filter.id_mask = 0x7F0;            // ID が“0x040-0x04F”のメッセージを承認
filter.min_dlc = 4;          // メッセージデータは 4 バイト以上必要
filter.label = 0x800;        // このタイプのメッセージに適用する任意のラベル


/* チャネル 0 に規則を追加。フィルタをかけたメッセージを受信メールボックス 3 に転送。*/
err = R_CAN_AddRxRule(CAN_CH0, &filter, CAN_BOX_RXMBX_3);
```

Example 2: メッセージの完全一致

```
can_filter_t  filter;
can_err_t     err;

/* フィルタを設定*/
filter.check_ide = 1;        // メッセージの IDE フィールドをチェック
filter.ide = 0;                    // 11 ビット ID
filter.check_rtr = 0;        // メッセージの RTR フィールドはチェックしない
filter.rtr = 0;                    // (チェックしないので、値は何でも良い)
filter.id = 0x040;           // メッセージ ID
filter.id_mask = 0x7FF;            // ID は“0x040”と完全に一致の必要あり
filter.min_dlc = 6;          // メッセージデータは 6 バイト以上必要
filter.label = 0x700;        // このタイプのメッセージに適用する任意のラベル


/* チャネル 0 に規則を追加。フィルタをかけたメッセージを受信メールボックス 2 に転送。*/
err = R_CAN_AddRxRule(CAN_CH0, &filter, CAN_BOX_RXMBX_2);
```

Special Notes:

通信モードに遷移後は規則に遷移できません。

R_CAN_Control()

特殊な動作とモードの変更を扱う関数です。

Format

```
can_err_t  R_CAN_Control(can_cmd_t  cmd,
                         uint32_t   arg1);
```

Parameters

cmd

実行するコマンドを指定

```
typedef enum e_can_cmd
{
  CAN_CMD_ABORT_TX,                 //引数: 送信メールボックス id
  CAN_CMD_RESET_TIMESTAMP,
  CAN_CMD_SET_MODE_COMM,            //通常のバス通信を開始
  CAN_CMD_SET_MODE_TST_STANDARD,
  CAN_CMD_SET_MODE_TST_LISTEN,
  CAN_CMD_SET_MODE_TST_EXT_LOOPBACK,
  CAN_CMD_SET_MODE_TST_INT_LOOPBACK,
  CAN_CMD_END_ENUM
} can_cmd_t;
```

argl

コマンド特定の引数。コマンドの多くは引数を取りません。

コマンド CAN_CMD_ABORT_TX の引数は送信メールボックス ID です（2.10.1 参照）。

Return Values

CAN_SUCCESS	*/*正常動作 */*
CAN_ERR_INVALID_ARG	*/*無効な引数が提供されました。*/*
CAN_ERR_ILLEGAL_MODE	*/*現在のモードから要求されるモードへの変更は不正です。*/*

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

本関数を使って、タイムスタンプカウンタのリセット、メールボックスのメッセージ送信の中止、CAN モードの変更が行えます。

以下の順番で関数を呼び出して CAN を設定します。

```
R_CAN_Open();
R_CAN_InitChan();
R_CAN_ConfigFIFO();   // FIFO は 0 以上
R_CAN_AddRxRule();    // 1～16 規則
```

設定後は、CAN モジュールはノーマル通信モードかテストモードに遷移する必要があります。
R_CAN_Control(); // CAN_CMD_SET_MODE_COMM、または CAN_CMD_SET_MODE_TST_xxx を使用

注: チャネルでバスオフが検出された場合、そのチャネルは Halt モードに遷移し、すべての通信を中止します。その場合、バスオフ復帰が検出されて、アプリケーションによって R_CAN_Control (CAN_CMD_SET_MODE_COMM)が呼び出されるまで、チャネルは再開始できません。

Reentrant
この関数は、再入可能（リエントラント）です。

Example: ノーマル通信モードに遷移する

```
    can_err_t  err;

    err = R_CAN_Control(CAN_CMD_SET_MODE_COMM, 0);
```

Example: 送信中止

```
    can_err_t  err;

    /* チャネル 0 でメールボックス 0 の送信を中止 */
err = R_CAN_Control(CAN_CMD_ABORT_TX, CAN_BOX_CH0_TXMBX_0);
```

Special Notes:
テストモードについて以下にまとめます。

- 標準テストモード: CRC テストを許可します。
- リッスンオンリモード: 通信速度の検出に使用します。このモードでは R_CAN_SendMsg()を呼び出すことはできません。
- 内部ループバックモード: チャネルに送信されたメッセージを受信メッセージとして扱い、同じチャネルで処理します。ここでは CAN の送受信機能は経由しません。
- 外部ループバックモード: 内部ループバックと同じですが、CAN の送受信機能が使用されます。

R_CAN_SendMsg()

送信メールボックス、または送信 FIFO にメッセージを配置します。

Format

```
can_err_t  R_CAN_SendMsg(can_box_t    box_id,
                         can_txmsg_t  *p_txmsg);
```

Parameters

box_id

送信ボックス ID（メールボックスまたは FIFO。2.10.1 参照）

p_txmsg

送信するメッセージへのポインタ

```
typedef struct st_can_txmsg
{
  uint8_t      ide;
  uint8_t      rtr;
  uint32_t     id;
  uint8_t      dlc;
  uint8_t      data[8];
  bool_t       one_shot;     // エラー時リトライしない; 送信メールボックスのみ
  bool_t       log_history;  // ログが必要な場合は“true”に設定
  uint8_t      label;        // 履歴 FIFO の場合は 8 ビットラベル
} can_txmsg_t;
```

Return Values

CAN_SUCCESS	*/*正常動作 */*
CAN_ERR_INVALID_ARG	*/*無効な引数が提供されました。*/*
CAN_ERR_BOX_FULL	*/*送信メールボックス、または送信 FIFO がフルです。*/*
CAN_ERR_ILLEGAL_MODE	*/*現在のモードでは、メッセージは送信できません。*/*

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

本関数は、送信メールボックス、または送信 FIFO にメッセージを配置します。送信メールボックスが保持できるメッセージ数は 1、送信 FIFO が保持できるメッセージ数は 4 です。それぞれに保持可能なメッセージのエントリが既に存在する場合、すぐに CAN_ERR_BOX_FULL を返します。box_id が送信メールボックスを示し、かつ割り込みが許可されていない場合（CAN_CFG_INT_MBX_TX_COMPLETE = 0）、メッセージが送信されるまでこの関数から復帰しません。割り込みが許可されている場合、またはメッセージの転送先が送信 FIFO の場合、本関数は送信レジスタにメッセージを配置した直後に復帰します。

Reentrant
この関数は、ボックスが異なる場合、再入可能（リエントラント）です。

Example:

```
can_txmsg_t   txmsg;
can_err_t     err;

/* メッセージの設定 */
txmsg.ide = 0;              // ID フィールドは 11 ビット
txmsg.rtr = 0;              // ローカルメッセージ
txmsg.id = 0x022;           // 通信先 ID
txmsg.dlc = 5;              // データ長
txmsg.data[0] = 'h';        // データ…
txmsg.data[1] = 'e';
txmsg.data[2] = 'l';
txmsg.data[3] = 'l';
txmsg.data[4] = 'o';
txmsg.one_shot = false;      // エラー時、通常のリトライを実施
txmsg.log_history = false;   // 履歴 FIFO のログを取らない
txmsg.label = 0;             // (メッセージのログを取らないのでラベルは無視)

/*
 * 送信メールボックス 2 にメッセージを配置
 * 送信完了割り込みが許可されていない場合、メッセージ送信後に関数は復帰する
 * (ここではエラーは発生しない前提)。
 */
err = R_CAN_SendMsg(CAN_BOX_CH0_TXMBX_2, &txmsg);
```

Special Notes:
なし

R_CAN_GetMsg()

受信メールボックス、または FIFO からメッセージを取得します。

Format

```
can_err_t  R_CAN_GetMsg(can_box_t    box_id,
                        can_rxmsg_t  *p_rxmsg);
```

Parameters

box_id

受信ボックス ID（メールボックスまたは FIFO。2.10.1 参照）

p_rxmsg

読み出すメッセージバッファへのポインタ

```
typedef struct st_can_rxmsg
{
  uint8_t   ide;
  uint8_t   rtr;
  uint32_t  id;
  uint8_t   dlc;
  uint8_t   data[8];
  uint16_t  label;       // 受信規則により 12 ビットラベル
  uint16_t  timestamp;
} can_rxmsg_t;
```

Return Values

CAN_SUCCESS	*/*正常動作 */*
CAN_ERR_CH_NO_INIT	*/*チャネルが初期化されていません。*/*
CAN_ERR_INVALID_ARG	*/*無効な引数が提供されました。*/*
CAN_ERR_BOX_EMPTY	*/*取得できるメッセージがありません。*/*

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

本関数は受信メールボックス、または FIFO からメッセージを読み出して、提供されたメッセージバッファに配置します。ボックスにメッセージがない場合、CAN_ERR_BOX_EMPTY を返して、本関数は復帰します。

Reentrant

この関数は、ボックスが異なる場合、再入可能（リエントラント）です。

Example:

```
can_rxmsg_t   rxmsg;
can_err_t     err;

/* メッセージが受信メールボックス 3 に入るのを待機 */
while (R_CAN_GetMsg(CAN_BOX_RXMBX_3, &rxmsg) == CAN_ERR_BOX_EMPTY)
    ;

/* rxmsg にメッセージ格納 */
```

Special Notes:

なし

R_CAN_GetHistoryEntry()

送信履歴 FIFO からログエントリを取得します。

Format

```
can_err_t  R_CAN_GetHistoryEntry(can_box_t     box_id,
                                 can_history_t  *p_entry);
```

Parameters

box_id

送信履歴 FIFO（2.10.1 参照）

p_entry

読み出すエントリバッファへのポインタ

```
typedef struct st_can_history
{
  can_box_t    box_id;   // メッセージを送信するボックス
  uint8_t      label;    // 送信履歴データ情報: 8 ビットラベル
} can_history_t;
```

Return Values

CAN_SUCCESS	*/*正常動作 */*
CAN_ERR_INVALID_ARG	*/*無効な引数が提供されました。*/*
CAN_ERR_BOX_EMPTY	*/*取得できるメッセージがありません。*/*

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

引数の構造体メンバ log_history が “TRUE”のとき、R_CAN_SendMsg()が呼び出されるごとに、エントリが履歴 FIFO に追加されます。本関数は、送信履歴 FIFO からログエントリを読み出し、提供されるエントリバッファに配置します。FIFO にエントリがない場合、CAN_ERR_BOX_EMPTY を返して本関数は復帰します。本機能は通常の動作では必要ありません。

Reentrant

この関数は、ボックスが異なる場合、再入可能（リエントラント）です。

Example:

```
    can_history_t   entry;
    can_err_t      err;

    /* 送信履歴 FIFO のすべてのエントリを処理 */
    while (R_CAN_GetMsg(CAN_BOX_HIST_FIFO, &entry) == CAN_SUCCESS)
    {
        /* 処理を記載 */
    }
```

Special Notes:

なし

R_CAN_GetStatusMask()

要求された状態に応じて 32 ビットマスクを返します。ビットの#define は“CAN_MASK_xxx”の形式を取ります。

Format

```
uint32_t  R_CAN_GetStatusMask(can_stat_t  type,
                              uint8_t    chan,
                              can_err_t   *p_err);
```

Parameters

type

復帰する状態を指定

```
typedef enum e_can_stat
{
  CAN_STAT_FIFO_EMPTY,
  CAN_STAT_FIFO_THRESHOLD,
  CAN_STAT_FIFO_OVFL,           // 読み出し後、ビットをリセット
  CAN_STAT_RXMBX_FULL,
  CAN_STAT_GLOBAL_ERR,          // 読み出し後、DLC エラービットをリセット
  CAN_STAT_CH_TXMBX_SENT,       // 読み出し後、ビットをリセット
  CAN_STAT_CH_TXMBX_ABORTED,    // 読み出し後、ビットをリセット
  CAN_STAT_CH_ERROR,            // 読み出し後、ビットをリセット
  CAN_STAT_END_ENUM
} can_stat_t;
```

chan

状態復帰するチャネルを指定（“0”にしてください）。この引数は CAN_STAT_CH_xxx 要求にのみ適用されます。

p_err

エラーの戻り値へのポインタ

CAN_SUCCESS */*正常動作 */*

CAN_ERR_INVALID_ARG */*無効な引数が提供されました。*/*

Return Values

32 ビットボックス、またはエラーマスク。エラーマスクのビット定義は CAN_MASK_xxx の形式を取ります。定義の詳細は 2.10.10 を参照してください。

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

本関数は、要求される状態を基準にマスクを返します。ビットマスクの形式はすべて CAN_MASK_xxx です（2.10.10 参照）。

Reentrant

この関数は、再入可能（リエントラント）です。

Example:

```
can_err_t   err;
can_rxmsg_t rxmsg;


/* 受信メールボックスにメッセージが入るのを待機 */
while (R_CAN_GetStatusMask(CAN_STAT_RXMBX_FULL, 0, &err) == 0)
    ;

/* 受信メールボックス 3 がフルかどうかを確認 */
if (R_CAN_GetStatusMask(CAN_STAT_RXMBX_FULL, 0, &err) & CAN_MASK_RXMBX_3)
{
    /* メッセージの取得 */
    R_CAN_GetMsg(CAN_BOX_RXMBX_3, &rxmsg);
}
```

Special Notes:

なし

R_CAN_GetCountFIFO()

FIFO 内の項目数を返します。

Format

```
uint32_t  R_CAN_GetCountFIFO(can_box_t  box_id,
                             can_err_t  *p_err);
```

Parameters

box_id

確認する FIFO を指定（2.10.1 参照）

p_err

エラーの戻り値へのポインタ

CAN_SUCCESS */*正常動作 */*

CAN_ERR_INVALID_ARG */*無効な引数が提供されました。*/*

Return Values

FIFO の項目数

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

本関数は、box_id で指定された FIFO の項目数を返します。本関数は通常の動作では必要ありません。

Reentrant

この関数は、再入可能（リエントラント）です。

Example:

```
uint32_t   cnt;
can_err_t  err;

/* チャネル 0 の履歴 FIFO のメッセージ数を判定 */
cnt = R_CAN_GetCountFIFO(CAN_BOX_CH1_HIST_FIFO, &err);
```

Special Notes:

使用する FIFO は任意です。

R_CAN_GetCountErr()

送信エラー数、または受信エラー数を返します。

Format

```
uint32_t  R_CAN_GetCountErr(can_count_t  type,
                            uint8_t        chan,
                            can_err_t      *p_err);
```

Parameters

type

復帰する状態を指定

```
typedef enum e_can_count
{
  CAN_COUNT_RX_ERR,
  CAN_COUNT_TX_ERR,
  CAN_STAT_END_ENUM
} can_count_t;
```

chan

エラー数を返す対象チャネルを指定（“0”にしてください）。

p_err

エラーの戻り値へのポインタ

*CAN_SUCCESS /*正常動作 */*

*CAN_ERR_INVALID_ARG /*無効な引数が提供されました。*/*

Return Values

検出されたエラー数

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

本関数は、要求されるカウントタイプでチャネルの送信エラー数、または受信エラー数を返します。

Reentrant

この関数は、再入可能（リエントラント）です。

Example:

```
uint32_t   rxcnt,txcnt;
can_err_t  err;

/* 検出されたエラー数を取得 */
rxcnt = R_CAN_GetCountErr(CAN_COUNT_RX_ERR, CAN_CH0, &err);
txcnt = R_CAN_GetCountErr(CAN_COUNT_TX_ERR, CAN_CH0, &err);
```

Special Notes:

本関数の使用は任意です。この関数を使って、ネットワークの状態、およびエラーパッシブ状態（128 エラー）、またはバスオフ状態（255 エラー）への遷移の可能性を検出できます。

R_CAN_Close()

CAN モジュールを終了し、関連する割り込みを無効にします。

Format

```
void  R_CAN_Close(void);
```

Parameters

なし

Return Values

なし

Properties

r_rscan_rx_if.h にプロトタイプ宣言されています。

Description

本関数は既存の通信をすべて停止し、すべての割り込みを無効にして、CAN モジュールを終了します。

Reentrant

この関数は、再入可能（リエントラント）です。ただし、1 度呼び出した後に、再度呼び出す必要はありません。

Example:

```
    R_CAN_Close();
```

Special Notes:

なし

R_CAN_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_CAN_GetVersion(void);
```

Parameters
なし

Return Values
バージョン番号

Properties
r_rscan_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。バージョン番号は符号化され、最上位の 2 バイトがメジャーバージョン番号を、最下位の 2 バイトがマイナーバージョン番号を示しています。

Reentrant
この関数は、再入可能（リエントラント）です。

Example
```
    uint32_t version;
    version = R_CAN_GetVersion();
```

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

4. 端子の設定

RSCAN FIT モジュールを使用するには、周辺機器機能の入出力信号を、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。このドキュメントでは、端子割り当てを「端子設定」と呼びます。R_CAN_Open 関数を呼び出した後、端子設定を実行してください。

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: rskrx231）です。例えば、RSKRX231 用の rscan FIT モジュールのデモプロジェクトは rscan_demo_rskrx231 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 rscan_demo_rskrx231.zip となります。

5.1 rscan_demo_rskrx231, rscan_demo_rskrx231_gcc

本プログラムには、メッセージの送受信が可能な CAN デバイス（スニファなど）の接続が要求されます。プログラムでは、1 つのメッセージを送信して受信する処理を一度に行います。受信されたデータの ID は "0x60-0x6F"で、データ長は 4 バイト以上の必要があります。

ビットレートは 500Kbps です。

本プログラムは、以下のいずれかで動作します。

- 割り込みなしで、メールボックスを使用
- 割り込みありで、FIFO を使用

動作の設定は main.c で USE_FIFOS の値を変更して行います。

- USE_FIFOS = 0: メールボックス
- USE_FIFOS = 1: FIFO

5.2 rscan_demo_rskrx24t, rscan_demo_rskrx24t_gcc

本プログラムには、メッセージの送受信が可能な CAN デバイス（スニファなど）の接続が要求されます。プログラムでは、1 つのメッセージを送信して受信する処理を一度に行います。受信されたデータの ID は "0x60-0x6F"で、データ長は 4 バイト以上の必要があります。

ビットレートは 500Kbps です。

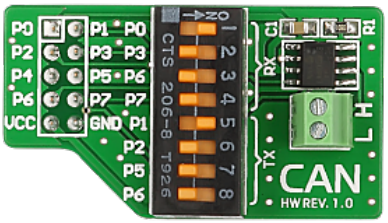
本プログラムは、以下のいずれかで動作します。

- 割り込みなしで、メールボックスを使用
- 割り込みありで、FIFO を使用

動作の設定は main.c で USE_FIFOS の値を変更して行います。

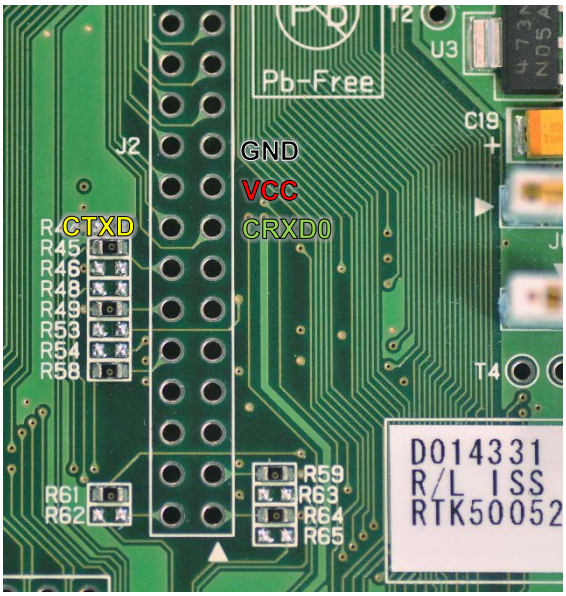
- USE_FIFOS = 0: メールボックス
- USE_FIFOS = 1: FIFO

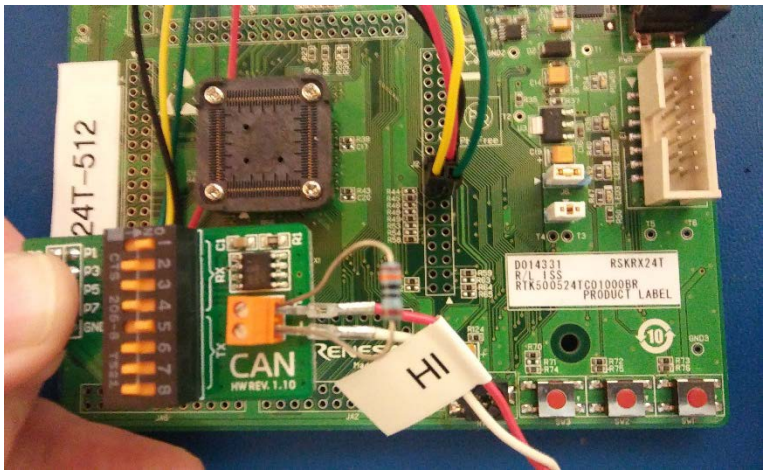
RSKRX24T が CAN モジュールに対応するには、ROM が大容量の RX24T (例: 512Kb EAxFP) を搭載する必要があります。また、外付けの CAN トランシーバボードも必要です。以下に MikroElektronika 社製の CAN-1 ボード (www.mikroe.com/add-on-boards/communication/can)を使用した例を示します。



2x3 ヘッダを RSKRX24T の J2 端子 15～20 にはんだ付けされることを推奨します。CAN-1 ボードのディップスイッチライン 1 と 5 は ON にしてください。ボードは以下のように繋いでください。

	RSKRX24T	CAN-1
CRXD0	J2 pin 15	P0
CTXD0	J2 pin 16	P1
VCC	J2 pin 17	VCC
GND	J2 pin 19	GND
120Ω の抵抗		P0-P1（ネットワークによっては終端抵抗を使用）





5.3 rscan_demo_rskrx24u, rscan_demo_rskrx24u_gcc

本プログラムには、メッセージの送受信が可能な CAN デバイス（スニファなど）の接続が要求されます。プログラムでは、1 つのメッセージを送信して受信する処理を一度に行います。受信されたデータの ID は "0x60-0x6F"で、データ長は 4 バイト以上の必要があります。

ビットレートは 500Kbps です。

本プログラムは、以下のいずれかで動作します。

- 割り込みなしで、メールボックスを使用
- 割り込みありで、FIFO を使用

動作の設定は main.c で USE_FIFOS の値を変更して行います。

- USE_FIFOS = 0: メールボックス
- USE_FIFOS = 1: FIFO

6. 付録

6.1 動作確認環境

このセクションでは、RSCAN 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.2.50)

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 V.2022-07 IAR Embedded Workbench for Renesas RX 4.20.03
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 V3.04.00 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 8.3.0.202202 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 4.20.3 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.2.50
使用ボード	Renesas Starter Kit+ for RX231 (型名.: R0K505231SxxxBE) Renesas Starter Kit+ for RX24U (型名.: RTK500524UCxxxxxBR)

表 6.2 動作確認環境 (Rev.2.40)

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.2022-01 IAR Embedded Workbench for Renesas RX 4.20.03
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 V3.04.00 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 8.3.0.202104 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 4.20.3 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.2.40
使用ボード	Renesas Starter Kit+ for RX140 (型名.: RTK55140xxxxxxxxxx)

表 6.3 動作確認環境 (Rev.2.30)

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.7.8.0
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 V3.02.00 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 8.3.0.201904 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 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。
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.2.30
使用ボード	Renesas Starter Kit+ for RX231 (型名.: RTK505231xxxxxxxx) Renesas Starter Kit+ for RX24U (型名.: RTK50524Uxxxxxxxx)

表 6.4 動作確認環境 (Rev.2.21)

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 V.7.7.0 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 V3.02.00 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 8.3.0.201904 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 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.2.21
使用ボード	Renesas Solution Starter Kit+ for RX23E-A (型名.: RTK0ESXBxxxxxxxxxx)

表 6.5 動作確認環境 (Rev.2.20)

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 V.7.7.0 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.2.20
使用ボード	Renesas Starter Kit+ for RX23E-A（型名 : RTK5523E-Axxxxxxxxx）

表 6.6 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
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
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 RX23W（型名：RTK5523Wxxxxxxxxx）

表 6.7 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 RX231（型名：R0K505231xxxxxx）

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)』

- 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 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_rscan_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_rscan_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_rscan_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、「2.8 コンパイル時の設定」を参照してください。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2016.10.31	—	初版発行
1.10	2017.02.17	—	FIT モジュールの RX230 グループ、RX24U グループ、RX24T グループ（ROM 512KB 版を含む）対応
		—	誤記修正
		5	2.5 対応ツールチェーン: 対応ツールチェーンのバージョンを更新
		16, 17	3.2 R_CAN_Open()、Special Notes: 追加。
		40-42	4.2 rscan_demo_rskrx24t、4.3 rscan_demo_rskrx24u: 追加
1.20	2018.12.06	ﾌﾟﾛｸﾞﾗﾑ	MCU がビッグエンディアンで動作していると正しく通信できない不具合を修正。 ■修正内容 内部の構造体・共用体定義に evenaccess を追加。
1.21	2019.02.01	ﾌﾟﾛｸﾞﾗﾑ	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。
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	「対象デバイス」RX24T グループから（ROM 512KB 版）を削除 「ターゲットコンパイラ」のセクションを追加。
		8	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
		41	「2.9 コードサイズ」セクションを更新。
		42	「4 端子設定」を追加。
		45	セクション「5. デモプロジェクト」を変更。
		45	「6. 付録」を追加。
		46	「6.1 動作確認環境」：Rev.2.00 に対応する表を追加。
			「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_CAN_GetVersion 関数のインライン展開を削除。 2. NOP を BSP の固有関数で置き換えた。 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	2019.06.28	1	RX23W のサポートを追加。
		9	RX23W に対応するコードサイズを追加。
		45	「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	RX23W のサポートを追加。
2.20	2019.10.15	1	RX23E-A のサポートを追加。
		9	RX23E-A に対応するコードサイズを追加。
		11, 24-25	check_ide と check_rtr のデータ型を変更。
		45	「6.1 動作確認環境」：Rev.2.20 に対応する表を追加。
		プログラム	RX23E-A のサポートを追加。 アトミック制御の処理を追加。 R_CAN_GetMsg()内の IAR ワーニングの修正。 check_ide と check_rtr のデータ型を変更。

2.21	2020.03.31	10 46	2.9 コードサイズのセクションを更新。 6.1 動作確認環境」：Rev.2.21 に対応する表を追加。 RX23E-A のサポートを追加。
2.30	2020.06.30	43,44,45 46 プログラム	デモプロジェクトの更新と追加 6.1 動作確認環境」：Rev.2.30 に対応する表を追加。 デモプロジェクトの更新と追加
2.31	2020.10.30	プログラム	XML ファイルの<zipsource>タグを修正
2.32	2021.09.13	20 プログラム	APN R01AN3455 への参照を削除。 デモプロジェクトに CS+ のサポートを追加。
2.40	2021.11.11	1, 4 10 46 プログラム	RX140 グループ (ROM 容量が 128K バイト以上の製品)を追加。 2.9 コードサイズのセクションを更新。 6.1 動作確認環境」：Rev.2.40 に対応する表を追加。 RX140 グループ (ROM 容量が 128K バイト以上の製品)を追加。
2.41	2022.06.30	23 プログラム	アプリケーションノートからマクロ 「CAN_ERR_MAX_ONE_GWFIFO」を削除 CAN クロックソースのデフォルト値を PCLK に変更しました。
2.50	2022.07.29	46 11 プログラム	6.1 動作確認環境」：Rev.2.50 に対応する表を追加。 RXMBX の列挙型を修正:CAN_BOX_RXMBX_4 削除して CAN_BOX_RXMBX_3 を追加しました。また、 CAN_BOX_RXMBX_2 の値を変更しました デモプロジェクトを更新 RXMBX の列挙型を修正:CAN_BOX_RXMBX_4 削除して CAN_BOX_RXMBX_3 を追加しました。また、 CAN_BOX_RXMBX_2 の値を変更しました

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。