RX ファミリ

CAN API Firmware Integration Technology

要旨

Renesas CAN API (Application Programming Interface)を使って、CAN バス上のデータを送信、受信、監視できます。本ドキュメントでは、CAN API の使用方法と CAN モジュールのいくつかの機能について説明します。

本アプリケーションノートでは CAN API のソースコードファイルが提供されます。API を使ったデモのソースコードも提供され、can_api_demo.c、および switches.c に記載されています。

対象デバイス

- RX64M グループ
- RX71M グループ
- RX65N グループ、RX651 グループ
- RX66T グループ
- RX72T 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合わせて変更し、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「9.1 動作確認環境 」を参照してください。

目次

1. 概要

RX CAN モジュールにはメールボックスが 32 個あり、CAN バス上で通信を行います。「メールボックス」とは、MCU の CAN モジュール内で実際にメッセージが格納される場所を指します。本書では「メールボックス」という用語を使用しますが、「メッセージボックス」、「メッセージバッファ」と呼ばれることもあります。メールボックスはメッセージを格納するバッファで、入ってきたデータで上書きされるか、MCU によって上書きされるまで、CAN のデータフレームを保持します。

各メールボックスは送信、または受信の設定が動的に行えます。多くの場合は受信に設定され、あまり送信に設定されることはありませんが、自由に設定して構いません。

1.1 基本情報

CAN は、安全性とリアルタイムな動作が優先されるアプリケーション向けに設計されていて、信頼性の高い通信手段を提供します。

CAN の通信はマルチマスタ、マルチスレーブが基本です。送信されるメッセージ、またはデータフレームには送信ノードのアドレスも、受信ノードのアドレスも含まれません。そのため、どのノードでもマスタ、またはスレーブになれます。メッセージの送信は、ブロードキャスト送信、または送信時にその ID をリッスンしていたノードへの送信が行えます。また、他のノードを更新することなく、新規のノードを追加できます。CAN はこのように柔軟に設計できることから、合理的、かつ冗長性を備え、再構築が容易なシステムを構築できます。

CAN の主な特性を以下に示します。

- 高い信頼性と耐ノイズ性
- チップでのエラー処理
- バス 2 線／ノード接続ポイントー配線コストの低減
- 柔軟なアーキテクチャ
- ネットワークの拡大を容易に実現

エラー処理はチップで行われるため、下層でのエラー処理を扱う複合スタックソフトウェアは必要ありません。MCU のバスコネクタで必要な端子は 2 端子のみのため、CAN ネットワークも物理層にあります。こうすることで、複数のバス接続が必要なネットワークスキームよりも高い信頼性を実現できます。新規のノードは、バス線上で追加したい個所をタップするだけで、簡単に追加できます。

接続可能なノード数とケーブル長がビットレートによって決定されます。有効な CAN データビットレートは 62.5Kbps、125Kbps、250Kbps、500Kbps、1Mbps です。ネットワークは、最高速度において、40 メートルのケーブルで 30 ノードに対応し、最低速度において、1000 メートルのケーブルで 100 ノード以上に対応します。

CAN ネットワークの基本構成は、CAN マイクロコントローラ (MCU) 、ファームウェア、バス信号の伝搬、および読み込みを行う CAN トランシーバ、バスメディア (2 線) から成ります。アプリケーションに応じて、十分なメールボックスを備えた CAN MCU を選択してください。

1.2 通信層

下図に CAN の通信層を示します。アプリケーション層が最上層、ハードウェア層が最下層となります。

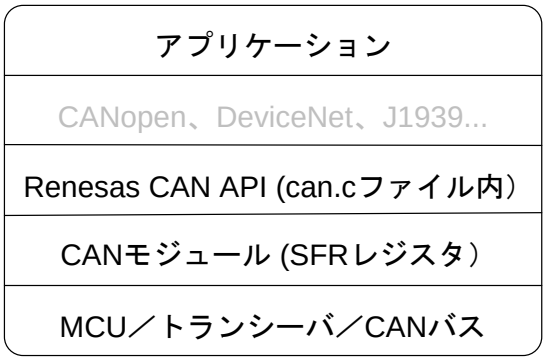

図 1 CAN の物理層とソースコード層

本ドキュメントでは、CANopen や DeviceNet などの上層のプロトコルには触れていません。(Renesas CAN MCU の中には、CANopen ソリューションに対応しているものがあります。詳しくは販売店にお問い合わせください)

1.3 接続

CAN モジュールのプロトコルコントローラは、CAN の送信端子 (CTXn)、および受信端子 (CRXn)を介して、外部バスのトランシーバに接続する必要があります。

1.4 メールボックス

CAN プロトコルコントローラは CAN モジュールのメールボックスに対して読み込みと書き込みを行います。CAN メッセージが送信されるとき、そのメッセージは、アプリケーションのファームウェアによってメールボックスに書き込まれます。その後、ID がより低いメッセージが他のノードから送信された場合を除いて、バスがアイドル状態になり次第、メッセージが自動的に送信されます。メールボックスが受信に設定されている場合、メッセージはプロトコルコントローラによってメールボックスに書き込まれます。メールボックスはネットワークからの次のメッセージに備えて空けておかなければならないので、ユーザは API を使ってこのメッセージをユーザメモリ領域にコピーする必要があります。

メールボックスの書き込みと読み出しは API によって行われます。ユーザはアプリケーションデータフレーム用の構造体を提供します。この構造体は、API 関数によって、入力メッセージの書き込みと、出力メッセージのコピーに使用されます。少なくとも出力メッセージと入力メッセージ用に各 1 つは構造体を作成することをお勧めします。出力メッセージ用は、ローカル変数（スタック上に配置）にすることも可能です。入力メッセージ用はメールボックスごとに作成することが推奨されます。この CAN データフレーム構造体 (can_frame_t) は API のヘッダファイルで提供します。以下に構造体を示します。

```
typedef struct
{
  uint32_t id;
  uint8_t dlc;
  uint8_t data[8];
} can_frame_t;
```

この構造体にはタイムスタンプは含まれませんが、その分、簡単に追加できます。

CAN バスのアービトレーションを除いて、メールボックス番号の低い方が優先されます。ただし、SH (RCAN-ET)ではメールボックス番号の高い方が優先されます。優先に関しては、送信および受信動作で共通しています。2 つのメールボックスが同じ CAN ID に設定された場合、メールボックス番号が低い方が優先されます。2 つのメールボックスが同じ ID で受信に設定された場合、一方のメールボックスのみがメッセージを受信します（もう一方は受信しません）。

1.5 拡張 CAN

拡張 ID を使用するには､r_can_rx_config.h で FRAME_ID_MODE を設定します｡拡張 CAN が有効な場合、‘Xid’が末尾に付く API 関数が呼び出されます。これらの関数では、CAN メールボックスの ID フィールドが自動的に拡張 ID 用にフォーマットされますので、‘Xid’関数を呼び出すだけで、構造体“can_frame_t”の持つ ID の値が 29 ビット（11 ビットではない）の ID として送信されます。

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。

- CAN モジュール（CAN）

2.2 ハードウェアリソースの要求

本 FIT モジュールに必要なハードウェアリソースについて説明します。明示的に記載していない限り、これらのリソースは他の周辺機能では使用できません。

2.2.1 周辺機能

CAN モジュール（CAN）

2.2.2 その他の周辺機能

CAN バスの送受信用に I / O ポートを割り当てる必要があります。 割り当てられたポートは汎用入出力ポートとして使用できません。

本 FIT モジュールでは、スタンバイ信号とイネーブル信号として汎用入出力ポートを使ってチャネルごとに制御するオプション機能があります。

2.3 ソフトウェアの要求

本 FIT モジュールは以下の FIT モジュールに依存しています。

- ボードサポートパッケージ (r_bsp) v5.20 以上

2.4 制限事項

本 FIT モジュールは FIFO メールボックスモードには対応していません。

2.5 サポートされているツールチェーン

本 FIT モジュールは「9.1 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_ can_rx_if.h に記載しています。
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_can_rx_config.h ファイルで選択または定義されています。

本 FIT モジュールの API をユーザプログラムから参照するには、r_can_rx_if.h をインクルードしてください。

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

アプリケーションで必要な機能性を満たすために、r_can_rx_config.h の変更が必要な場合があります。例えば、CAN ポーリングモード、または CAN 割り込みモードで実行する場合は変更が必要です。ルネサス CAN API の関数が定義されている r_can_rx.c への変更は前提としていませんが、API で提供されない機能を追加することで機能を向上できる場合もあります。

e2 studio の Smart Configurator を使用してこのソフトウェアをインストールする場合、この FIT モジュールの設定は Smart Configurator の 「コンポーネント」→「プロパティ」で行います。

それ以外の場合、以降の表を参考にして r_can_rx_config.h を手動で編集します。

2.8.1 割り込みモードとポーリングモードおよび割り込み優先レベル

送受信されたメッセージの CAN メールボックスのチェック方法を設定します。

割り込みモードに設定する場合、使用チャネルの割り込み優先レベルも設定します。

定義	設定値	説明
USE_CAN_POLL	0 = 割り込みモード 1 = ポーリングモード	送受信されたメッセージの CAN メールボックスをチェックする方法を設定します。
CAN0_INT_LVL	0 ～ 15 (0 のとき割り込み禁止)	チャネル 0 の割り込み優先レベルを設定します。
CAN1_INT_LVL	0 ～ 15 (0 のとき割り込み禁止)	チャネル 1 の割り込み優先レベルを設定します。
CAN2_INT_LVL	0 ～ 15 (0 のとき割り込み禁止)	チャネル 2 の割り込み優先レベルを設定します。

2.8.2 標準 ID と拡張 ID

本 FIT モジュールで有効にする CAN ID のタイプを、11 ビットの標準 ID か 29 ビットの拡張 ID から選択します。設定オプションは、STD_ID_MODE、EXT_ID_MODE、または MIXED_ID_MODE に設定できます。MIXED_ID_MODE に設定した場合、すべての API を使用できます。

バスに標準と拡張の両フレームが存在する場合は MIXED_ID_MODE を使用してください。他の ID モードを選択すると、正しいデータが得られないことがあります。

定義	設定値	説明
FRAME_ID_MODE	STD_ID_MODE = 標準(11 ビット) CAN ID. EXT_ID_MODE = 拡張(29 ビット) CAN ID. MIXED_ID_MODE = 標準(11 ビット) と拡張(29 ビット) ID の両方を使用	STD_ID_MODE または EXT_ID_MODE は、その ID モードに属する API 関数のみを有効にします。MIXED_ID_MODE に設定すると、全ての API が使用できます。

2.8.3 チャネルの有効化と端子のマッピング

各チャネルを有効にすることで、ビルド対象とすることができます。チャネルを無効にすると、無効にしたチャネルに対応するコードがビルドから除外されます。

CAN の送信および受信端子の設定は、端子設定、ポート・モード、端子の方向などに影響します。特定の端子のみが、CAN の送信および受信端子に使用できます。

また、MCU に接続される CAN トランシーバの制御端子も設定が必要です。これらの端子は CAN モジュール専用ではないため、汎用入出力端子を使って設定できます。トランシーバの中にはこの他にも制御端子を持つものがあり、それらを使用する場合は設定が必要です。

定義	設定値	説明
CAN_USE_CAN0	0 = 無効 1 = 有効	チャネル 0 を有効または無効にします。
CAN0_RX_PORT	RX71M, RX64M, RX65N の場合 P33, PD2 RX72T, RX66T の場合 P22, PA1, PA7, PB6, PC6, PE0, PF3	チャネル 0 の受信端子として割り当てることができる端子です。 デバイスのパッケージによって使用可能な端子は異なります。
CAN0_TX_PORT	RX71M, RX64M, RX65N の場合 P32, PD1 RX72T, RX66T の場合 P23, PA0, PA6, PB5, PC5, PD7, PF2	チャネル 0 の送信端子として割り当てることができる端子です。 デバイスのパッケージによって使用可能な端子は異なります。
CAN_USE_CAN0_STANDBY_ENABLE_PINS	0 = 無効 1 = 有効	チャネル 0 におけるスタンバイ端子とイネーブル端子を有効または無効にします。
CAN0_TRX_STB_PORT	ポートグループ名	スタンバイ端子に使うポートのポートグループ名
CAN0_TRX_STB_PIN	ポート番号	スタンバイ端子に使うポートのポート番号
CAN0_TRX_STB_LVL	0 = Low アクティブ 1 = High アクティブ	スタンバイ信号のアクティブレベル
CAN0_TRX_ENABLE_PORT	ポートグループ名	イネーブル端子に使うポートのポートグループ名
CAN0_TRX_ENABLE_PIN	ポート番号	イネーブル端子に使うポートのポート番号
CAN0_TRX_ENABLE_LVL	0 = Low アクティブ 1 = High アクティブ	イネーブル信号のアクティブレベル
CAN_USE_CAN1	0 = 無効 1 = 有効	チャネル 1 におけるスタンバイ端子とイネーブル端子を有効または無効にします。
CAN1_RX_PORT	RX71M, RX64M, RX65N の場合 P15, P55	チャネル 1 の受信端子として割り当てることができる端子です。
CAN1_TX_PORT	RX71M, RX64M, RX65N の場合 P14, P54	チャネル 1 の送信端子として割り当てることができる端子です。

定義	設定値	説明
CAN_USE_CAN1_STANDBY_ENABLE_PINS	0 = 無効 1 = 有効	チャネル 1 におけるスタンバイ端子とイネーブル端子を有効または無効にします。
CAN1_TRX_STB_PORT	ポートグループ名	スタンバイ端子に使うポートのポートグループ名
CAN1_TRX_STB_PIN	ポート番号	スタンバイ端子に使うポートのポート番号
CAN1_TRX_STB_LVL	0 = Low アクティブ 1 = High アクティブ	スタンバイ信号のアクティブレベル
CAN1_TRX_ENABLE_PORT	ポートグループ名	イネーブル端子に使うポートのポートグループ名
CAN1_TRX_ENABLE_PIN	ポート番号	イネーブル端子に使うポートのポート番号
CAN1_TRX_ENABLE_LVL	0 = Low アクティブ 1 = High アクティブ	イネーブル信号のアクティブレベル
CAN_USE_CAN2	0 = 無効 1 = 有効	チャネル 2 におけるスタンバイ端子とイネーブル端子を有効または無効にします。
CAN2_RX_PORT	RX71M, RX64M の場合 P67	チャネル 2 の受信端子として割り当てることができる端子です。
CAN2_TX_PORT	RX71M, RX64M の場合 P66	チャネル 2 の送信端子として割り当てることができる端子です。
CAN_USE_CAN2_STANDBY_ENABLE_PINS	0 = 無効 1 = 有効	チャネル 2 におけるスタンバイ端子とイネーブル端子を有効または無効にします。
CAN2_TRX_STB_PORT	ポートグループ名	スタンバイ端子に使うポートのポートグループ名
CAN2_TRX_STB_PIN	ポート番号	スタンバイ端子に使うポートのポート番号
CAN2_TRX_STB_LVL	0 = Low アクティブ 1 = High アクティブ	スタンバイ信号のアクティブレベル
CAN2_TRX_ENABLE_PORT	ポートグループ名	イネーブル端子に使うポートのポートグループ名
CAN2_TRX_ENABLE_PIN	ポート番号	イネーブル端子に使うポートのポート番号
CAN2_TRX_ENABLE_LVL	0 = Low アクティブ 1 = High アクティブ	イネーブル信号のアクティブレベル

2.8.4 ビットレート設定

ビットレートの設定は、以降で説明する「R_CAN_SetBitrate」の説明と r_can_rx_config.h ファイルを参照してください。

定義	設定値	説明
CAN_BRP	ユーザーズマニュアル ハードウェア編を参照	プリスケーラ分周比選択ビット（BCR.BRP）の設定値
CAN_TSEG1	4 ～ 16	タイムセグメント 1 制御ビット（BCR.TSEG1）の設定値
CAN_TSEG2	2 ～ 8	タイムセグメント 2 制御ビット（BCR.TSEG2）の設定値
CAN_SJW	1 ～ 4	再同期ジャンプ幅制御ビット（BCR.SJW）の設定値

2.8.5 レジスタの最大ポーリング時間

CAN レジスタのビットが期待値を得たかどうかをポーリングするときの最大ループ回数。ポーリングモードを使用する場合、メールボックスがフレームを受信したことを確認するために一定時間待ちたい場合、この値を増加してください。また、小さい値も設定できますが、“0”は設定しないでください。“0”に設定した場合、メールボックスは全く確認されません。

定義	設定値	説明
MAX_CANREG_POLLCYCLES	0 より大きい整数	ポーリングモードにのみ有効です。 CAN レジスタのビットが期待値を得たかどうかをポーリングするときの最大ループ回数を設定します。

2.9 コードサイズ

コードサイズの前提は、最適化レベル（optimization level）2 に設定し、Renesas CCRX toolchain 3.01、GCC for Renesas RX 4.8.4、および IAR Embedded Workbench for Renesas RX 4.10.1 を使用することです。ROM (コード、定数、事前初期化済みデータ) と RAM (事前初期化済みデータ、未初期化データ) のサイズは、デバイスに対応するモジュール設定ヘッダ参照ファイル内で設定したビルド時の設定オプションによって決まります。

ROM と RAM の使用量

ビルド設定	領域	サイズ (バイト)		
		CCRX	GCC	IAR
ポーリングモード、チャネル 0 のみが有効 CAN0 Standby/Enable 端子が未使用	ROM	2497	5044	3413
割り込みモード、チャネル 0 のみが有効 CAN0 Standby/Enable 端子が未使用	ROM	2940	5612	3973
割り込みモード、3 個のチャネルが有効 すべての CAN Standby/Enable 端子が有効	ROM	2982	5780	3983
すべて	RAM	36	36	36

2.10 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) 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。

3. API 関数

API を使用することによって、細かい設定を気にせずに CAN モジュールを使用でき、ユーザアプリケーションとネットワーク上のノード間での通信が簡単に行えます。

CAN の設定と通信は CAN レジスタを使って行います。詳細はお使いの MCU のユーザーズマニュアル ハードウェア編を参照してください。通信を成立させるには、CAN モジュールのレジスタを正しい順番で設定して、読み出す必要があります。CAN API を使えば、このような作業が簡単に行えます。

CAN モジュールの初期設定後に必要なのは、受信および送信に使用する API 呼び出しのみです。その後は定期的に CAN のエラー状態を確認します。エラー状態になった場合、アプリケーションは待機して、CAN モジュールの復帰を監視します。CAN モジュールはエラーの状態に応じてオンライン、オフラインになります。CAN モジュールの復帰が確認できたら、アプリケーションを再スタートします。

3.1 関数一覧

本 FIT モジュールには以下の API 関数があります。

関数名	説明
R_CAN_Create()	CAN モジュールを初期化します。
R_CAN_PortSet()	MCU とトランシーバのポート端子を設定します。
R_CAN_Control()	CAN の動作モードを設定します。
R_CAN_SetBitrate()	CAN のビットレート（通信速度）を設定します。
R_CAN_TxSet()、R_CAN_TxSetXid()	メールボックスを送信に設定します。
R_CAN_Tx()	CAN バスへのメッセージ送信を開始します。
R_CAN_TxCheck()	データフレームが正常に送信されたことを確認します。
R_CAN_TxStopMsg()	フレーム送信を要求されたメールボックスを停止します。
R_CAN_RxSet()、R_CAN_RxSetXid()	メールボックスを受信に設定します。
R_CAN_RxPoll()	メールボックスに受信メッセージがあるかどうかを確認します。
R_CAN_RxRead()	メールボックスから CAN データフレームの内容を読み出します。
R_CAN_RxSetMask()	CAN ID の承認マスクを設定します。
R_CAN_CheckErr()	CAN モジュールのバスおよびエラーの状態を確認します。

3.2 戻り値

API に関連する戻り値	説明
R_CAN_OK	処理が正常に完了しました。
R_CAN_NOT_OK	処理に失敗しました。通常は各エラーに特定の戻り値が返されます。
R_CAN_SW_BAD_MBX	不正なメールボックス番号です。
R_CAN_BAD_CH_NR	存在しないチャネル番号です。
R_CAN_BAD_ACTION_TYPE	この関数では対応していないアクションです。
R_CAN_MSGLOST	メッセージが上書きされたか、失われました。
R_CAN_NO_SENTDATA	メッセージは送信されませんでした。
R_CAN_RXPOLL_TMO	受信メッセージのポーリングが時間切れです。
R_CAN_SW_WAKEUP_ERR	CAN モジュールがスリープモードから復帰しません。
R_CAN_SW_SLEEP_ERR	CAN モジュールがスリープモードに遷移しませんでした。
R_CAN_SW_HALT_ERR	CAN モジュールが Halt モードに遷移しませんでした。
R_CAN_SW_RST_ERR	CAN モジュールがリセットモードに遷移しませんでした。
R_CAN_SW_TSRC_ERR	タイムスタンプエラー
R_CAN_SW_SET_TX_TMO	前の送信完了待ちが時間切れです。

API に関連する戻り値	説明
R_CAN_SW_SET_RX_TMO	前の受信完了待ちが時間切れです。
R_CAN_SW_ABORT_ERR	アボート処理待ちが時間切れです。
R_CAN_MODULE_STOP_ERR	CAN モジュールがモジュールストップ状態（低消費電力）です。

CAN バス状態に関連する戻り値	バスの状態
R_CAN_STATUS_ERROR_ACTIVE	通常動作
R_CAN_STATUS_ERROR_PASSIVE	ノードは送信エラーカウンタ、または受信エラーカウンタについて、127 を超えるエラーフレームを送信しました。
R_CAN_STATUS_BUSOFF	ノードの送信失敗により､エラーカウンタが 255 を超えています。

3.3 R_CAN_Create

CAN モジュールの初期設定を行います。ユーザ通信のコールバック関数、CAN 割り込み、およびビットレートを設定し、メールボックスのデフォルト設定を行って、CAN オペレーションモードに遷移します。

本関数で、CAN 割り込みの優先レベルとユーザコールバック関数を設定します。本関数は R_CAN_SetBitrate()も呼び出し、マスクをデフォルト（フレームをマスクしない）に設定します。

Format

```
uint32_t R_CAN_Create(const uint32_t  ch_nr,
                      void (*tx_cb_func)(void),
                      void (*rx_cb_func)(void),
                      void (*err_cb_func)(void))
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

tx_cb_func

メールボックスの送信完了時、CAN API から呼び出されるユーザアプリケーションの関数名。
ポーリングモードを使用する場合、または割り込みモードでコールバック関数を使用しない場合、NULL を指定します。

rx_cb_func

メールボックスの受信完了時、CAN API から呼び出されるユーザアプリケーションの関数名。
ポーリングモードを使用する場合、または割り込みモードでコールバック関数を使用しない場合、NULL を指定します。

err_cb_func

CAN エラー発生時、CAN API から呼び出されるユーザアプリケーションの関数名。
ポーリングモードを使用する場合、または割り込みモードでコールバック関数を使用しない場合、NULL を指定します。

Return Values

R_CAN_OK	*処理が正常に完了しました。*
R_CAN_SW_BAD_MBX	*不正なメールボックス番号です。*
R_CAN_BAD_CH_NR	*存在しないチャネル番号です。*
R_CAN_SW_RST_ERR	*CAN モジュールがリセットモードに遷移しませんでした。*
R_CAN_MODULE_STOP_ERR	*CAN モジュールがモジュールストップ状態（低消費電力）です。PRCR レジスタでモジュールストップ状態が解除されていないようです。*

R_CAN_Control()関数の戻り値もご確認ください。

Properties

r_can_rx_if.h にプロトタイプ宣言されています。r_can_rx.c で実装されます。

Description

本関数は CAN モジュールを CAN スリープモードから復帰させて、CAN リセットモードに遷移させます。また、メールボックスを下記のデフォルト設定に設定します。

- 新規フレーム到着時は、メールボックスの読み出し未のデータは上書きする。
- デバイスを ID 優先送信モード（メールボックス番号優先モードではなく、CAN の通常動作）に設定する。
- すべてのメールボックスのマスクを無効にする。

r_can_rx_config.h で USE_CAN_POLL がコメント化されている場合、R_CAN_SetBitrate 関数を呼び出して、CAN 割り込みを設定します。

本関数は復帰する前にすべてのメールボックスをクリアし、CAN モジュールをオペレーションモードに設定し、エラーをクリアします。

Example

```
#if USE_CAN_POLL
  api_status = R_CAN_Create(g_can_channel, NULL, NULL, NULL);
#else
  /* 割り込みを使用 */
  api_status = R_CAN_Create(g_can_channel,
                            my_can_tx0_callback,
                            my_can_rx0_callback,
                            my_can_err0_callback);
#endif
```

3.4 R_CAN_PortSet

MCU とトランシーバのポート端子を設定します。

“Enable”などのトランシーバのポート端子は設計により異なりますので、その内容に応じて本関数でも修正が必要です。

また、本関数を使って、リッスンオンリモードなどの CAN ポートテストモードへの遷移も可能です。

Format

```
uint32_t     R_CAN_PortSet(const uint32_t     ch_nr,
                           const uint32_t     action_type   );
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

action_type

ポートのアクション

ENABLE :CAN ポート端子と CAN トランシーバを有効にします。

DISABLE :CAN ポート端子と CAN トランシーバを無効にします。

CANPORT_TEST_LISTEN_ONLY :リッスンオンリモードに設定します。
ACK またはエラーフレームは送信されません。
「5.2 リッスンオンリ（バスモニタ）」を参照してください。

CANPORT_TEST_0_EXT_LOOPBACK:外部バスおよびループバックを使用します。
これは、初回のデバッグ時に有用です。
「5.1 ループバック」を参照してください。

CANPORT_TEST_1_INT_LOOPBACK :メールボックスとの通信を内部でのみ行います。
これは、初回のデバッグ時に有用です。
「5.1 ループバック」を参照してください。

CANPORT_RETURN_TO_NORMAL : ポートを通常の使用に戻します。

Return Values

R_CAN_OK	*処理が正常に完了しました*
R_CAN_SW_BAD_MBX	*不正なメールボックス番号です。*
R_CAN_BAD_CH_NR	*存在しないチャネル番号です。*
R_CAN_BAD_ACTION_TYPE	*この関数では対応していないアクションです。*
R_CAN_SW_HALT_ERR	*CAN モジュールが Halt モードに遷移しませんでした。*
R_CAN_SW_RST_ERR	*CAN モジュールがリセットモードに遷移しませんでした。*

R_CAN_Control()関数の戻り値もご確認ください。

Properties

r_can_rx_if.h にプロトタイプ宣言されています。r_can_rx.c で実装されます。

Description

ループバックモードを使用している場合（初回のテストまたはデバッグ時）を除いては、ボードのデフォルト設定を行う関数（例: hwsetup）を呼び出した後に、本関数を呼び出してください。

MCU の CAN ポート端子が他のボードセットアップコード（r_bsp の設定）で設定された場合、それらの端子からの不正な High/Low 出力がバスに悪影響を与えていないか注意してください。あるノードのハードリセットによって、他のノードがエラーモードになることがあります。これは、CAN がポートを再設定する前に、すべてのポートの High/Low 出力をデフォルトで設定したためと考えられます。このような問題を引き起こすコードは削除する必要があります。このようなコードがあると、わずかな期間、ポートは High/Low 信号を出力し、CAN バスの電圧レベルを狂わせる可能性があります。

お使いのトランシーバに応じて、必要があれば、トランシーバのポート端子を変更、または追加してください。

Example:

```
/* CAN バスの通常使用 */
R_CAN_PortSet(0, ENABLE);
```

3.5 R_CAN_Control

CAN の動作モードを設定します。

CAN 制御レジスタで指定された CAN 動作モードへの遷移を制御します。例えば、Halt モードは、後に受信メールボックスを設定するために使用されます。

Format

```
uint32_t R_CAN_Control(    const uint32_t      ch_nr,
                           const uint32_t      action_type   );
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

action_type

CAN モジュールのアクション

EXITSLEEP_CANMODE : CAN スリープモードから復帰します。
スリープモードは CAN モジュール開始時のデフォルトモードです（「7. CAN スリープモード」参照）。

ENTERSLEEP_CANMODE : CAN スリープモードに遷移します。
このモードでは、消費電力が低減されます。

RESET_CANMODE : CAN モジュールをリセットモードに遷移させます。

HALT_CANMODE : CAN モジュールを Halt モードに遷移させます。
CAN モジュールはバスに接続された状態ですが、通信は停止されます。

OPERATE_CANMODE : CAN モジュールをオペレーションモードに遷移させます。

Return Values

R_CAN_OK	*処理が正常に完了しました*
R_CAN_SW_BAD_MBX	*不正なメールボックス番号です。*
R_CAN_BAD_CH_NR	*存在しないチャネル番号です。*
R_CAN_BAD_ACTION_TYPE	*この関数では対応していないアクションです。*
R_CAN_SW_WAKEUP_ERR	*CAN モジュールがスリープモードから復帰しません。*
R_CAN_SW_SLEEP_ERR	*CAN モジュールがスリープモードに遷移しませんでした。*
R_CAN_SW_HALT_ERR	*CAN モジュールが Halt モードに遷移しませんでした。*
R_CAN_SW_RST_ERR	*CAN モジュールがリセットモードに遷移しませんでした。*

R_CAN_PortSet()関数の戻り値もご確認ください。

Properties

r_can_rx_if.h にプロトタイプ宣言されています。

Description

Halt モードへ遷移するために本 API を呼び出す場合を除いて、CAN のモード遷移は、他の API 関数を介して自動的に行われます。例えば、開始時のデフォルトのモードは CAN スリープモードです。他の動作モードには API を使って切り替えます。例えば、ビットレートと割り込みの設定に使用する CAN レジスタを初期化する場合、スリープモードから復帰してリセットモードに遷移します。その後、Halt モードに遷移して、メールボックスを設定します。

Example:

```
/* CAN バスの通常使用 */
result = R_CAN_Control(0, OPERATE_CANMODE); //結果が“R_CAN_OK”であることを確認。
```

3.6 R_CAN_SetBitrate

CAN のビットレート（通信速度）を設定します。

CAN の設定時はビットレートとビットタイミングを必ず設定する必要があります。なお、リセットモードに遷移すれば、これらの設定は後から変更できます。

Format

```
void R_CAN_SetBitrate(const uint32_t ch_nr);
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

Return Values

なし

Properties

r_can_rx_if.h にプロトタイプ宣言されています。

Description

CAN バスのビットレート、またはデータ速度を設定するには、ユーザーズマニュアル ハードウェア編の図表を参照いただいた上で、CAN のビットタイミングと MCU 周波数に関する理解が要求されます。API では、ビットレートのデフォルトは 500KB です。MCU クロック、または周辺クロックの周波数が変更されない限りは、デフォルト設定で関数を呼び出すだけで動作します。

ビットレートは r_can_rx_config.h 内のマクロで設定します。マクロを設定するためには、いくつかの計算が必要です。CAN システムクロック(f_{canclk})は CAN 周辺クロックの内部クロック周期です。この CAN システムクロックは CAN のボーレートプリスケーラ値および周辺バスクロックによって決定されます。1Tq は CAN クロック周期と等しくなります。

CAN バスの 1 ビット時間は複数の Tq の総和です。各ビットレートレジスタには、CAN の 1 ビット周期（Tqtot）を構成する Tq の総数が設定されます。

ビットレートレジスタを設定するための計算式

PCLK は周辺クロック周波数、PCLKB です。

fcan = PCLK/EXTAL

プリスケーラ値によって CAN 周辺クロックの周波数を下げます。

fcanclk = fcan/prescaler

1 Tq は CAN クロックの 1 周期です。

Tq =1/fcanclk

Tqtot は、CAN の 1 ビット時間内の CAN 周辺クロック周期の総数で、「時間セグメント」と「SS（常に “1”）」の合計で構成される周辺クロックによって成り立ちます。

コードでの Tqtot は以下のようになります。

(BSP_CFG_XTAL_HZ * BSP_CFG_PLL_MUL) / (CAN_BRP * BITRATE * BSP_CFG_PCKB_DIV)

これらのマクロを設定して、Tqtot が CAN レジスタで許容されている数値より大きくならないようにします。ユーザーズマニュアル ハードウェア編でビットレート設定例の表を参照してください。

その他の制限を以下に示します。

Tqtot = TSEG1 + TSEG2 + SS (条件: TSEG1 > TSEG2)

SS は常に“1”です。多くの場合、同期ジャンプ幅 (SJW)はバス・アドミニストレータによって提供されます。“1 ≦ SJW ≦ 4”を選択します。

詳細は r_can_rx_config.h をご覧ください。

ビットレートの変更には、以下の Python コードを利用することもできます。

```
# Python 3.5.1. Python のシンプルなコードを利用して、ビットレートレジスタの設定値を計算する。
Python を持っていない場合も以下のコードを追ってみてください。レジスタの設定値をマニュアルで計算する
方法を確認できます。

from fractions import Fraction
BITRATE = 500000

# BRP 試行。レジスタ設定に対して TQTOT が大きすぎる場合は数値を上げる。
CAN_BRP = 4

# TQTOT が完全な整数でない場合、許容範囲内に制限してください。
# そうでない場合は正確なビットレートを取得できず、
# 数値はテストできません。
MAX_TQ_FRACTION_DEV = 0.1

XTAL_HZ = 12000000
PLL_MUL = 4 # MCU によってはこの定義は使用不可の場合あり。その場合は“1”に設定。
PCKB_DIV = 2
TQTOT = (XTAL_HZ * PLL_MUL)/(CAN_BRP * BITRATE * PCKB_DIV)

print ("TQTOT is", round(TQTOT, 2), "=> Set TSEG1 larger than TSEG2, and SJW
to 1, so that the sum of these is TQTOT.")
print ("==============")
```

Example:

```
/* r_can_rx_config.h で定義されたとおりにビットレートを設定 */
R_CAN_SetBitrate(0);
```

3.7 R_CAN_TxSet、R_CAN_TxSetXid

メールボックスを送信に設定します。

R_CAN_TxSet は、指定された ID、データ長、データフレームペイロードをメールボックスに書き込み、R_CAN_Tx()を呼び出して、メールボックスを送信モードに設定し、フレームをバスに送信します。

R_CAN_TxSetXid は、R_CAN_TxSet と同様の動作ですが、ID が 29 ビット ID になります。

Format

```
uint32_t   R_CAN_TxSet(  const uint32_t          ch_nr,
                         const uint32_t          mbox_nr,
                         const can_frame_t*      frame_p,
                         const uint32_t          frame_type        );

uint32_t   R_CAN_TxSetXid(     const uint32_t          ch_nr,
                               const uint32_t          mbox_nr,
                                     can_frame_t*      frame_p,
                               const uint32_t          frame_type        );
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

mbox_nr

使用するメールボックス（0 ～ 32）

frame_p*

メモリ内のデータフレーム構造体へのポインタ。

この構造体には、送信されるデータフレームを構成する ID、DLC、およびデータが含まれます。

frame_type

DATA_FRAME: 通常のデータフレームを送信

REMOTE_FRAME: リモートフレームの要求を送信

Return Values

R_CAN_OK	*メールボックスが送信に設定されました。*
R_CAN_SW_BAD_MBX	*不正なメールボックス番号です。*
R_CAN_BAD_CH_NR	*存在しないチャネル番号です。*
R_CAN_BAD_ACTION_TYPE	*この関数では対応していないアクションです。*

Properties

r_can_rx_if.h にプロトタイプ宣言されています。

Description

本関数は、まずは指定されたメールボックスで以前の送信が完了するのを待ちます。その後、メールボックスの割り込みを一時的に無効にして、メールボックスにメールボックスの ID 値、および frame_p で示されるデータ長コードを設定し、データフレームかリモートフレームかを選択し、最後にメールボックスにデータフレームペイロードバイト (0～7)をコピーします。USE_CAN_POLL が定義されている場合を除いて、メールボックスの割り込みを有効に戻します。R_CAN_Tx を呼び出して、メッセージを送信します。

Example:

```
#define MY_TX_SLOT 7
can_frame_t      my_tx_dataframe;
my_tx_dataframe.id = 1;
my_tx_dataframe.dlc = 2;
my_tx_dataframe.data[0] = 0xAA;
my_tx_dataframe.data[1] = 0xBB;

/* フレーム送信 */
api_status = R_CAN_TxSet(0, MY_TX_SLOT, &my_tx_dataframe, DATA_FRAME);
```

3.8 R_CAN_Tx

CAN バスへのメッセージ送信を開始します。

本 API はメールボックスが前のフレームの処理を完了するまで待ってから、メールボックスを送信モードに設定します。

Format

```
uint32_t  R_CAN_Tx(  const uint32_t      ch_nr,
                     const uint32_t      mbox_nr      );
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

mbox_nr

使用するメールボックス（0 ～ 32）

Return Values

R_CAN_OK	*送信設定が正常に行われました。*
R_CAN_SW_BAD_MBX	*不正なメールボックス番号です。*
R_CAN_BAD_CH_NR	*存在しないチャネル番号です。*
R_CAN_SW_SET_TX_TMO	*前の送信完了待ちが時間切れです。*
R_CAN_SW_SET_RX_TMO	*前の受信完了待ちが時間切れです。*

Properties

r_can_rx_if.h にプロトタイプ宣言されています。

Description

本関数はメールボックスの内容を送信するだけですので、システムがメールボックスの内容の設定を開始してから、少なくとも一度は R_CAN_TxSet を呼び出す必要があります。

Example:

```
#define MY_TX_SLOT 7

/* メールボックスの内容を送信。メールボックスはこれより以前に送信に設定されていることが前提。*/
R_CAN_Tx(0, MY_TX_SLOT);
```

3.9 R_CAN_TxCheck

データフレームが正常に送信されたことを確認します。

Format

```
uint32_t  R_CAN_TxCheck( const uint32_t    ch_nr,
                         const uint32_t    mbox_nr     );
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

mbox_nr

使用するメールボックス（0 ～ 32）

Return Values

R_CAN_OK	*送信が正常に完了しました。*
R_CAN_SW_BAD_MBX	*不正なメールボックス番号です。*
R_CAN_BAD_CH_NR	*存在しないチャネル番号です。*
R_CAN_MSGLOST	*メッセージが上書きされたか、失われました。*
R_CAN_NO_SENTDATA	*メッセージは送信されませんでした。*

Properties

r_can_rx_if.h にプロトタイプ宣言されています。

Description

本関数は、アプリケーションでメッセージの送信を確認する必要がある場合にのみ使用します。例えば、ステートマシンの処理を実行したい場合や、連続でメッセージを送信したい場合などに使用します。チップ上の CAN の通信制御であれば、API によってメールボックスの送信が実行された場合、かなりの確度でメールの送信は実行されたと言えます。送信をより確実にしたい場合、送信後に本関数をご使用ください。

Example:

```
/*** 対象のフレームが送信されたことを確認 */
api_status = R_CAN_TxCheck(0, CANBOX_TX);

if (api_status == R_CAN_OK)
{
  /* メインアプリケーションに通知 */
  message_x_sent_flag = TRUE;
}
```

3.10 R_CAN_TxStopMsg

フレーム送信を要求されたメールボックスを停止します。

Format

```
uint32_t R_CAN_TxStopMsg(  const uint32_t      ch_nr,
                           const uint32_t      mbox_nr      );
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

mbox_nr

使用するメールボックス（0 ～ 32）

Return Values

R_CAN_OK	*処理が正常に完了しました。*
R_CAN_SW_BAD_MBX	*不正なメールボックス番号です。*
R_CAN_BAD_CH_NR	*存在しないチャネル番号です。*
R_CAN_SW_ABORT_ERR	*アボート処理待ちが時間切れです。*

Properties

r_can_rx_if.h にプロトタイプ宣言されています。

Description

本関数はメールボックス制御フラグをクリアして、送信を停止します（TrmReq を“0”に設定）。このとき、ソフトウェアカウンタで、最大期間までアボート処理を待機します。

メッセージ送信が停止しなかった場合、R_CAN_SW_ABORT_ERR を返します。このエラーの原因としては、メッセージが送信済みだったことが考えられます。

Example:

```
R_CAN_TxStopMsg(0, MY_TX_SLOT);
```

3.11 R_CAN_RxSet、R_CAN_RxSetXid

メールボックスを受信に設定します。

R_CAN_RxSet は、指定された CAN ID を持つデータフレームを受信するようにメールボックスを設定します。その ID を持つデータフレームがメールボックスに格納されます。

R_CAN_RxSetXid は、R_CAN_RxSet と同様の動作ですが、ID が 29 ビット ID になります。

Format

```
uint32_t       R_CAN_RxSet(      const uint32_t      ch_nr,
                                 const uint32_t      mbox_nr,
                                 const uint32_t      sid,
                                 const uint32_t      frame_type    );

uint32_t       R_CAN_RxSetXid(   const uint32_t      ch_nr,
                                 const uint32_t      mbox_nr,
                                       uint32_t      xid,
                                 const uint32_t      frame_type    );
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

mbox_nr

使用するメールボックス（0 ～ 32）

sid

xid

メールボックスが受信する CAN ID（0 ～ 7FFh）

frame_type

DATA_FRAME :通常のデータフレームを送信

REMOTE_FRAME :リモートフレームの要求を送信

Return Values

R_CAN_OK	*処理が正常に完了しました。*
R_CAN_SW_BAD_MBX	*不正なメールボックス番号です。*
R_CAN_BAD_CH_NR	*存在しないチャネル番号です。*
R_CAN_SW_SET_TX_TMO	*前の送信完了待ちが時間切れです。*
R_CAN_SW_SET_RX_TMO	*前の受信完了待ちが時間切れです。*

Properties

r_can_rx_if.h にプロトタイプ宣言されています。

Description

本関数は、まずは指定されたメールボックスで以前の送信／受信が完了するのを待ちます。その後、メールボックスの割り込みを一時的に無効にして、メールボックスに指定された標準 ID を設定し、通常のデータフレーム、またはリモートフレーム要求のいずれを受信するかを設定します。

Example:

```
#define MY_RX_SLOT      8
#define SID_FAN_SPEED   0x10

R_CAN_RxSet(0, MY_RX_SLOT, SID_FAN_SPEED, DATA_FRAME);
```

3.12 R_CAN_RxPoll

メールボックスに受信メッセージがあるかどうかを確認します。

Format

```
uint32_t  R_CAN_RxPoll(  const uint32_t    ch_nr,
                         const uint32_t    mbox_nr     );
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

mbox_nr

確認するメールボックス（0 ～ 32）

Return Values

R_CAN_OK	*待機中のメッセージがあります。*
R_CAN_NOT_OK	*待機中、または保留中のメッセージはありません。*
R_CAN_RXPOLL_TMO	*保留中のメッセージがありますが、時間切れです。*
R_CAN_SW_BAD_MBX	*不正なメールボックス番号です。*
R_CAN_BAD_CH_NR	*存在しないチャネル番号です。*

Properties

r_can_rx_if.h にプロトタイプ宣言されています。

Description

指定のメッセージを受信するようにメールボックスを設定してから、その受信が正常に完了したことを確認することが重要です。確認は、以下の 2 つの方法で行えます。

1. ポーリングを使用。API を定期的に呼び出して、新規メッセージを確認します。この方法では、CAN の設定ファイルで USE_CAN_POLL を定義する必要があります。メッセージがあると判定された場合、R_CAN_RxRead を使って、メッセージを取得します。
2. CAN 受信割り込みを使用（USE_CAN_POLL は定義しない）。本 API を使って受信したメールボックスを確認し、結果をアプリケーションに通知します。

メールボックスに新規データが確認された場合、本関数は“R_CAN_OK”を返します。

Example:

R_CAN_RxRead()の Example を参照してください。

3.13 R_CAN_RxRead

メールボックスから CAN データフレームの内容を読み出します。

本 API は、指定されたメールボックスに受信メッセージがあることを確認します。メッセージが確認できた場合、メールボックスのデータフレームのコピーを該当する構造体に書き込みます。

Format

```
uint32_t  R_CAN_RxRead( const uint32_t          ch_nr,
                        const uint32_t          mbox_nr,
                        can_frame_t* const      frame_p);
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

mbox_nr

確認するメールボックス（0 ～ 32）

frame_p

メモリ内のデータフレーム構造体へのポインタを参照。

メールボックスが受信した CAN データフレームのコピーが配置されるデータ構造体へのアドレス

Return Values

R_CAN_OK	*待機中のメッセージがあります。*
R_CAN_SW_BAD_MBX	*不正なメールボックス番号です。*
R_CAN_BAD_CH_NR	*存在しないチャネル番号です。*
R_CAN_MSGLOST	*メッセージが上書きされたか、失われました。*

Properties

r_can_rx_if.h にプロトタイプ宣言されています。

Description

最初に R_CAN_RxPoll()を使って、メールボックスがメッセージを受信したかどうかを確認します。

ポーリングモード、または CAN 受信割り込み使用時、本関数を使って、メールボックスからメッセージを取得します。

Example:

```
#define MY_RX_SLOT 8
can_frame_t my_rx_dataframe;

api_status = R_CAN_RxPoll(0, CANBOX_RX_DIAG);
if (api_status == R_CAN_OK)
  R_CAN_RxRead(0, CANBOX_RX_DIAG, &my_rx_dataframe);
```

3.14 R_CAN_RxSetMask

CAN ID の承認マスクを設定します。

1 つの ID のみを承認するには、すべてのマスクを“1”に設定します。すべてのメッセージを承認する場合、すべてのマスクを“0”に設定します。ある範囲のメッセージを承認する場合、その範囲に対応する ID ビットを“0”に設定します。

Format

```
void R_CAN_RxSetMask(   const uint32_t    ch_nr,
                        const uint32_t    mbox_nr,
                        const uint32_t    mask_value  );
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

mbox_nr

マスクするメールボックス（0 ～ 32）。グループ内の 4 つのメールボックスに影響。

mask_value

マスク値（0 ～ 7FFh）

Return Values

なし

Properties

r_can_rx_if.h にプロトタイプ宣言されています。

Description

受信メールボックスはマスクを使って、1 つのメッセージを抽出することも、また、ある範囲のメッセージ（CAN ID の範囲)を受信することもできます。これは、メールボックスグループの ID フィールドを使って行われます。マスクはメールボックス 0～3 に 1 つ、4～7 に 1 つなどとなっています。**ですから、マスクを変更すると、隣接するメールボックスの動作に影響します。**

- マスクを“0”に設定することは、「このビットをマスクする」または「このビットは見ない」ということを意味し、ビットの内容に関わらず承認します。
- マスクを“1”に設定すると、その位置の CAN-ID ビットがメールボックスの CAN-ID と一致するかを確認します。

マスクの設定方法

メールボックスで受信したい CAN-ID の範囲を 700-704h とします。標準 11 ビット ID を使用する場合、該当範囲の ID は 16 進数とバイナリで以下のようになります。

16 進数	バイナリ
0x700	011100000000b
0x701	011100000001b
0x702	011100000010b
0x703	011100000011b
0x704	011100000100b

通常、メールボックスは設定された受信 ID と一致した ID を持つフレームのみを承認しますが、ビット位置のマスクが“0”の場合、0 と 1 の両方の ID ビットを承認します。その後、上記のすべてを承認したい場合、マスクを“011111111000b”、または“07F8h”としてマスクを設定します。

CAN 受信フィルタはビット位置 b10 (MSB)～b3 (LSB)のみを確認し、これらがメールボックスの受信 ID と一致しているかどうかを確認します。

その後、上記のマスクに属するいずれかのメールボックス（1 つのマスクごとに 4 つのメールボックスにグループ化）が ID 0x700 を受信するように設定した場合、そのメールボックスは 0x700～0x707 からすべての ID を承認します（ID を 0x700～0x707 に設定すると結果は同じ）。そのため、ID 0x705～0x707 は無視されるように、アプリケーションソフトウェアで設定する必要があります。

アクセプタンスフィルタサポートによるメッセージの高速フィルタリング

マスクを使って広範囲のメッセージ ID を受信した場合、ファームウェアを使って、実際に必要なメッセージをフィルタする必要があります。この検索速度を上げるために、アクセプタンスフィルタサポートを使用できます。

アクセプタンスフィルタサポートユニット (ASU)は、マスクを使ったメッセージのソフトウェアフィルタ（R_CAN_RxSetMask API 使用）と比べて、検索が高速で行えます。標準 ID ビットはメモリに通常のワードとして格納されず、再配置されるため、時間を要することがあります。また、承認マスクは、必要なメッセージを特定の組み合わせで受信できないとう点が問題になる場合があります。すべてのメッセージを承認するようにマスクを設定した場合、各入力 ID に対して、ソフトウェアを使って多数のメッセージを確認することで、不要な時間を費やしてしまうことがあります。また、この手動フィルタでは、すべての ID を読み込み可能なフォーマットで持たなくてはなりません。このような場合に効果的なソリューションが ASU です。

ASU を使用する場合、メッセージボックスに保存されているとおりに CAN-ID を ASU に書きます。ASU レジスタから読み戻すとき、ワード単位でテーブルを検索します。読み出したデータは次のような内容になっています。ビット 0～7 の構成は、アドレス検索情報 (ASI)で SID10～3 です。ビット 8～15 の構成は、ビット検索情報“BSI”です。SID0～3 はビット位置に変換されて、高速なテーブル検索を可能にします。

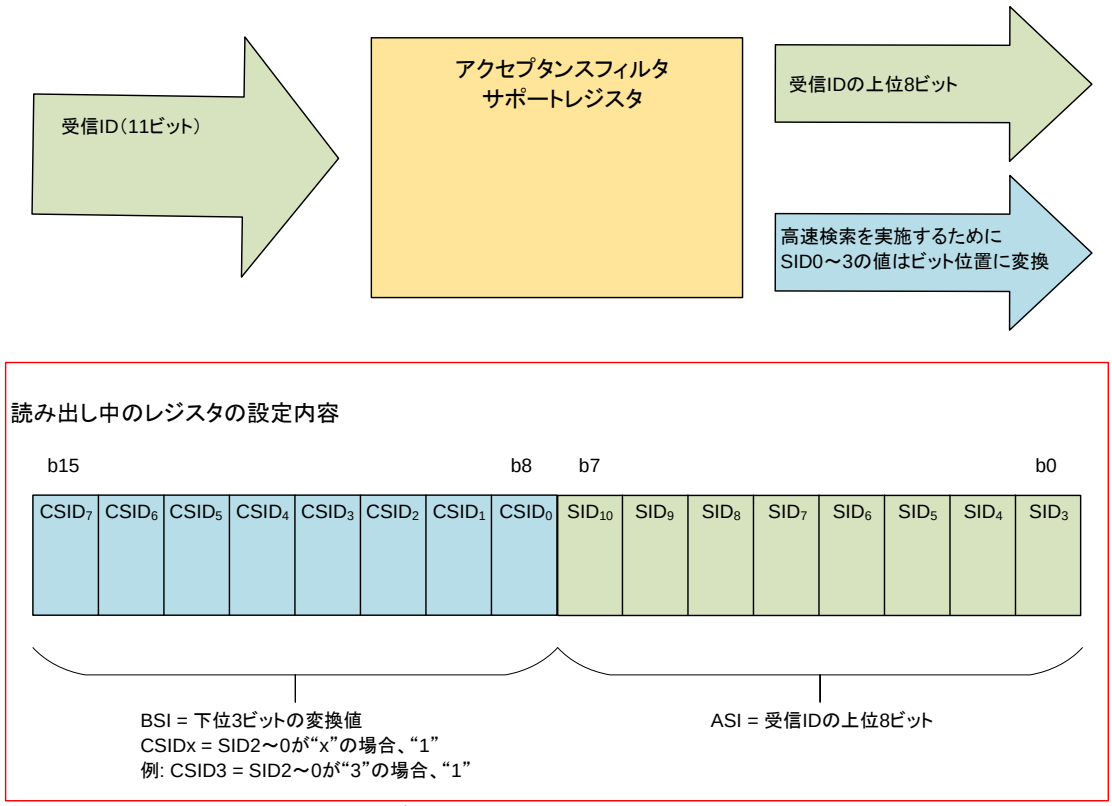

図 2 アクセプタンスフィルタサポートユニット (ASU)

読み出し時、テーブルの高速検索を可能にするために、ID がフォーマットされます。これによって、通常の CAN ID の配列を検索するよりも応答が速くなります。

検索テーブル

検索テーブルはユーザが用意する必要があり、アプリケーションで要求している ID かどうかを確認するために使用します。ファームウェアによって、各バイトの ASI および各ビットの BSI でテーブルを検索します。ビットの BSI 値がユーザのテーブルに設定され、ビットパターンがレジスタの BSI パターンと一致すると、そのアドレスはノードが要求する情報であることを意味し、アプリケーションによってフレームが処理されます。

3.15 R_CAN_CheckErr

CAN モジュールのバスおよびエラーの状態を確認します。

Format

```
uint32_t  R_CAN_CheckErr(const uint32_t  ch_nr);
```

Parameters

ch_nr

使用する CAN チャネル（0 ～ 2）（使用可能なチャネルは MCU に依存します）。

Return Values

R_CAN_ BAD_CH_NR	*存在しないチャネル番号*
R_CAN_STATE_ERROR_ACTIVE	*CAN バスの状態: 通常動作*
R_CAN_STATE_ERROR_PASSIVE	*CAN バスの状態:ノードは送信エラーカウンタ、または受信エラーカウンタについて、127 を超えるエラーフレームを送信しました。*
R_CAN_STATUS_BUSOFF	*CAN バスの状態: ノードの送信失敗により、エラーカウンタが 255 を超えています。*

Properties

r_can_rx_if.h にプロトタイプ宣言されています。

Description

本 API は CAN モジュールの CAN 状態フラグを確認し、状態エラーコードを返します。これによって、ノードが機能している状態かどうかを確認でき、アプリケーションのエラー処理に使用できます。

メインループから定期的にポーリングするか、CAN エラー割り込みを使用します。CAN モジュールは再送信とエラーフレームの処理を自動的に行うため、エラー割り込み処理は特に必要ありません。

エラー状態になった場合、CAN モジュールはエラーの状態に応じてオンライン、またはオフラインになりますので、アプリケーションは待機しながら、CAN モジュールの復帰を監視します。CAN モジュールの復帰が確認できたら、アプリケーションを再スタートします。

バスの状態

CAN は、CAN ネットワークのノードで異常が発生した場合、ネットワーク通信を保護するように設計されています。送信でエラーフラグが検出された場合、送信エラーカウンタがカウントアップされ、受信フレームでエラーが検出された場合、受信エラーカウンタがカウントアップされます。送信、および受信エラーカウンタはフレームが正常に送信、または受信される度に、それぞれカウントダウンされます。エラーアクティブ状態（通常動作の状態）、およびエラーパッシブ状態のときは、メッセージの送信および受信が行えます。

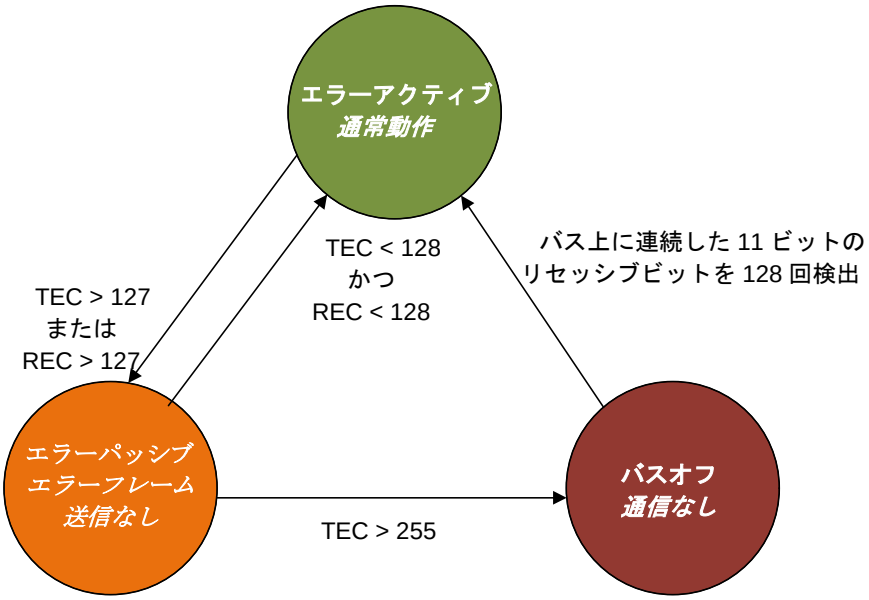

図 3　CAN バスのエラー状態

エラーアクティブ

ノードがエラーアクティブ状態の場合、バスと通常の通信を行っています。エラーが検出された場合、アクティブエラーフラグが送信されます。エラーカウントが 127 を超えたら、エラーパッシブ状態に切り替わります。

エラーパッシブ

送信、または受信エラーカウンタが 127 を超えた場合、そのノードの状態ははエラーパッシブ状態に変わります。この状態でもメッセージの送受信は行えますが、ノードはエラーフレームを送信しません。エラーフレームはユーザからは見えず、MCU の CAN モジュールによって処理されます。

バスオフ

送信エラーカウンタが 255 を超えた場合、CAN のノードはバスオフ状態になります。これによって、不具合ノードによってバスで障害が発生するのを防ぎます。深刻な問題によって、CAN ノードがバスオフ状態になった場合、バス上に連続した 11 ビットのリセッシブ(recessive)ビットが 128 回検出されるまで、または CAN モジュールがリセットされるまで、そのノードでメッセージの送信および受信は行えません。アプリケーションによってバスオフからの復帰が検出されたら、CAN モジュールのすべてのレジスタを初期化して、アプリケーションを再スタートする必要があります。

CAN のポーリングを使用する

ノードがバスオフ状態のときに通信が行われないように、API を定期的に呼び出して CAN の状態を確認します。以下の説明はメインアプリケーションのループごとに handle_can_bus_state()を 1 回呼び出した場合です。

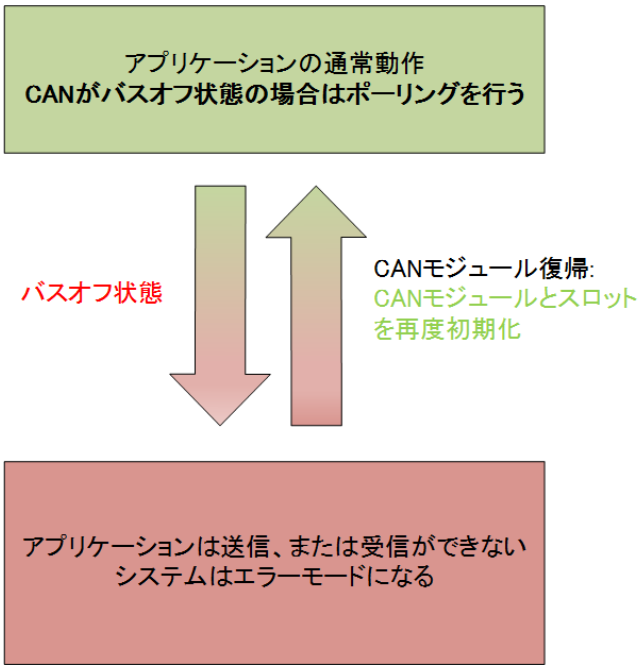

図 4　アプリケーションのバスオフ復帰の対応（バスオフ復帰は MCU で検出）

バス上で連続した 11 ビットのリセッシブビットを 128 回検出した後、ノードは通常のエラーアクティブ状態に自動復帰します。ノードがバスオフ状態になる時間は、1 ミリ秒以下など、非常に短い時間です。

メインルーチンのサイクルごとに、チェックエラー関数でポーリングを行うか、または CAN エラー割り込みを使って、ノードの状態を確認します。ノードが、一定期間内で一定回数バスオフ状態になった場合、警告を送信したり、LED を点灯することもできます。

バスオフになった場合にノードに要求される最低限のアクションを上図に示します。通信の試行を停止し、チェックエラー関数を使って、CAN モジュールが通常のエラーアクティブ状態に復帰しているかどうかを確認します。ノードが復帰してから、CAN モジュールとアプリケーションの初期設定を行い、スロットを適切な状態にします。

Example:

can_api_demo.c で handle_can_bus_state()を参照してください。

CAN エラー割り込みを使用する

CAN エラー割り込みを使って、ノードのエラー状態を確認できます。ただし、単純なエラーは CAN モジュールで処理されますので、通常は定期的にポーリングすれば十分です。

本 API はエラーISR から呼び出されると、エラー状態を判定し、状態遷移が発生したかどうかをアプリケーションに通知します。多くの場合、ここでの処理は、送信、または受信エラーカウンタがインクリメントされるのみです。

単一のエラー発生、エラーパッシブ状態への遷移、バスオフ状態への遷移のそれぞれに対して個別に割り込みを有効にできます。例えば、CAN エラー割り込みを有効にした場合、エラーが検出される度に割り込みが生成されます。ただし、CAN でエラー処理を行うため、通常は割り込みを生成する必要はありません。

4. デモプロジェクト

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。本 FIT モジュールには以下のデモプロジェクトが含まれます。

4.1 ワークスペースにデモを追加する

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「完了」をクリックします。

CAN アプリケーションデモコードのファイルは、..¥src ディレクトリにある can_api_demo.c、および switches.c です。

デモを実行するには、以下の説明に従って、圧縮 e² studio プロジェクト（can_demo_xxxx.zip）を e² studio にインポートします。

4.1.1 e² studio でプロジェクトをインポートしてデバッグする

1. 新規にワークスペースを作成する

1-1. ワークスペースを作成したい場所に空フォルダを作成します。

1-2. e² studio を開始し、ワークスペースとして、上記で作成したフォルダを指定します。

1-3. Workbench アイコン（「ようこそ」ウィンドウの右下）をクリックします。

以下の手順を続けます。

2. 既存のワークスペースを使用する

2-1. 「インポート」を選択します。

2-2. 「一般」→「既存プロジェクトをワークスペースへ」を選択します。または、アーカイブファイルかディレクトリから新しいプロジェクトを作成します。

- デモのコードがエクスポートして作成されたアーカイブ ZIP ファイルの場合、そのファイルを参照します。
- デモのコードがソースコード（.project ファイル）と一緒に e² studio プロジェクトのディレクトリにある場合、プロジェクトのルートディレクトリを参照します。コードをワークスペース（.metadata ディレクトリがある場所）に持ちたい場合、「プロジェクトをワークスペースにコピーする」を選択してください。

2-3. [終了]ボタンをクリックします。

2-4. アーティファクト名を$(ProjName)に変更します。「プロジェクト」→「プロパティ」→「C/C++ ビルド」→「設定」を選択します。こうすることで、プロジェクト名を変更した場合も、正しくビルドされます。

これでワークスペースにデモプロジェクトがインポートできました。同じワークスペースに別のプロジェクトをインポートすることもできます。

コードを実行する

デバッグセッションを作成してダウンロードし、コードを実行します。

4.1.2 デモを実行する

同梱の CAN API のデモプロジェクトには、CAN API を使って 500kbps で送受信を行うプログラムが含まれます。メールボックスのポーリング、または CAN 送信／受信割り込みのいずれを使っても、デモを実行できます。

デモはいくつかの方法で設定できます。

- 2 つのボードをプログラムし、CAN バスでそれらを接続します。デモをプログラムして実行する前に、一方のボードで、CAN ID 値“TX_CANID_DEMO_INIT”および“RX_CANID_DEMO_INIT”を切り替えます。
- CAN バスモニタ（例: SysTec 製　低コストモニタ 3204000）を使用して、デモでフレームを送受信します。
- R_CAN_PortSet API の CANPORT_TEST_1_INT_LOOPBACK を使って、内部で通信できます。外部バスは必要ありません。
- CAN 割り込みが有効な場合、リモートフレームのデモも行えます。

動作説明

デモはデフォルトの CAN-ID の TX_CANID_DEMO_INIT と RX_CANID_DEMO_INIT を使って、フレームを送受信します。テストフレーム NR_STARTUP_TEST_FRAMES をできるだけ高速で連続送信することによって、デモが開始されます。このデモの目的は、1) バスリンクを確認、2) メッセージを高速で連続送信する、の 2 点です。

ユーザアクション

SW1 を押下して CAN フレームを 1 つ送信します。TxID をインクリメントさせるには、SW2 を押した状態で、SW3 を押します。デモのアクションは、can_int_demo()関数、または can_poll_demo()関数内（いずれの関数かは r_can_rx_config.h の USE_CAN_POLL の設定による）で確認していただくのが一番わかりやすいです。

リモートフレーム

標準 CAN フレームの送受信の他に、デモプログラムは、CAN-ID 50h（can_api_demo.h の REMOTE_TEST_ID で定義）でメールボックスが受信したリモートフレームの要求に対する応答も送信できます。

この機能をデモに追加するには、can_api_demo.c で REMOTE_DEMO_ENABLE を“1”に設定します。

また、割り込みモードが要求されますので、CAN API の config ファイルで USE_CAN_POLL を“0”に設定します。リモートフレームの要求は、前述の CAN モニタなど、外部のソースから行う必要があります。リモートフレーム要求を CAN-ID 50h に送信するように、外部の CAN ソースを設定してください。

4.2 Renesas デバッグコンソール

E1/E20 から e² studio のデバッグコンソールに対してトレースデータを有効にすると、ユーザアプリケーションからリアルタイムでデータを出力することができます。これによって、C 言語の printf()を使って、トレースした文字列を送信して、標準出力が可能になります。この場合、標準出力は E1/E20 デバッグレジスタになります。

これを行うには、../r_config/r_bsp_config.h の BSP_CFG_IO_LIB_ENABLE を“1”に設定します。

デバッグコンソールを有効にするために、マクロが自動的にコードを有効にします。そのためには以下の手順を行ってください。

1. INIT_IOLIB()が呼び出されていることを、resetprog.c で確認してください。
2. lowlvl.c 内のコードには charput および charget 関数を含む必要があります。これによって、最下レベルの入出力処理に E1/E20 デバッグレジスタが使用されます。
   例えば、charput には以下を含む必要があります。

```
/* 送信バッファが空になるのを待機 */
while(0 != (E1_DBG_PORT.DBGSTAT & TXFL0EN));
```

3. printf を使用したい場合、ファイルに<stdio.h>を記載してください。
   printf()を呼び出すファイルには以下を追加します。
```
#if BSP_CFG_IO_LIB_ENABLE
    #include <stdio.h>
#endif
```

4. e² studio にて、以下のように[Renesas デバッグ仮想コンソールの有効化／無効化]および[コンソールのピン留め]の両方をクリックして、「デバッグコンソール」ウィンドウを追加します。**E1/E20 のプリントバッファを空にし、また、コードの実行がブロックされないようにするには、これらをオンにする必要があります。**

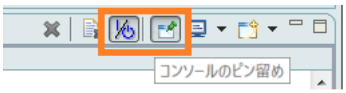

図 5 デバッグコンソールの制御ボタン。

コンソールが応答していないようであれば、e² studio で [Renesas デバッグ仮想コンソールの有効化／無効化]を再度押してください。

5. 何もプリントされない場合、[Renesas デバッグ仮想コンソールのクリア]（[Renesas デバッグ仮想コンソールの有効化／無効化]アイコンの左のアイコン）を数回押します。

5. テストモード

製品開発時などに有用なテストモードがあります。テストモードには、内部／外部ループバックモードと、リッスンオンリモードがあります。

5.1 ループバック

ループバックモードで、メールボックスが同じメッセージを受信するように設定すると、ノードが送信したメッセージをそのノードで受信します。これはアプリケーションをテストするのに、またアプリケーションのデバッグ中に自己診断するのに有用です。

5.1.1 内部ループバック: CAN バスを介さずにノードをテストする

内部ループバックモード、いわゆるセルフテストモードでは、バスに接続せずに、CAN メールボックスを介して通信が行えます。ノードは、データフレームの ACK ビットを使って送信したデータを認識します。また、メールボックスに同じ CAN ID が設定されていた場合、送信したデータを受信メールボックスに格納します。このような動作は通常の動作では行われません。

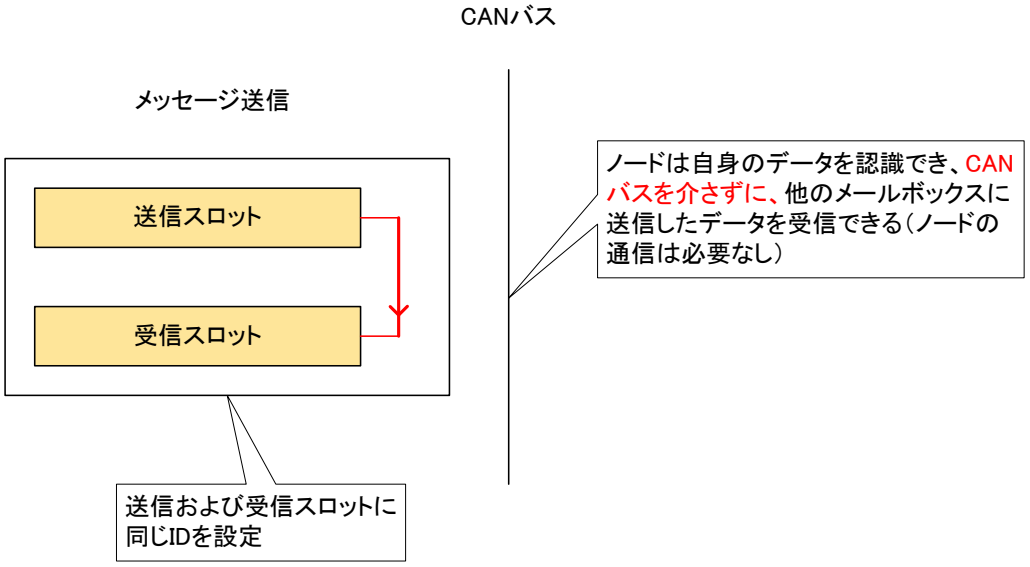

図 6 CAN 内部ループバックモード: CAN バスを介さないノードの機能テスト

内部ループバックはテスト時に有用です。内部ループバックでは、CAN コントローラは、バス上にノードが 1 つのときに ACK 未受信による CAN エラーを送信せずに動作できるため、送信したフレームを同じノードで受信します。

5.1.2 外部ループバック: テストノード

外部ループバックは、ノードが CAN バスと接続されていて、メッセージがバスに送信されるという点を除いて、内部ループバックと同じです。内部ループバックと同様に、ノードは送信したメッセージを認識しますので、ノードはバス上に 1 つで構いません。ノードを単独でテストできることは、この方法の利点です。

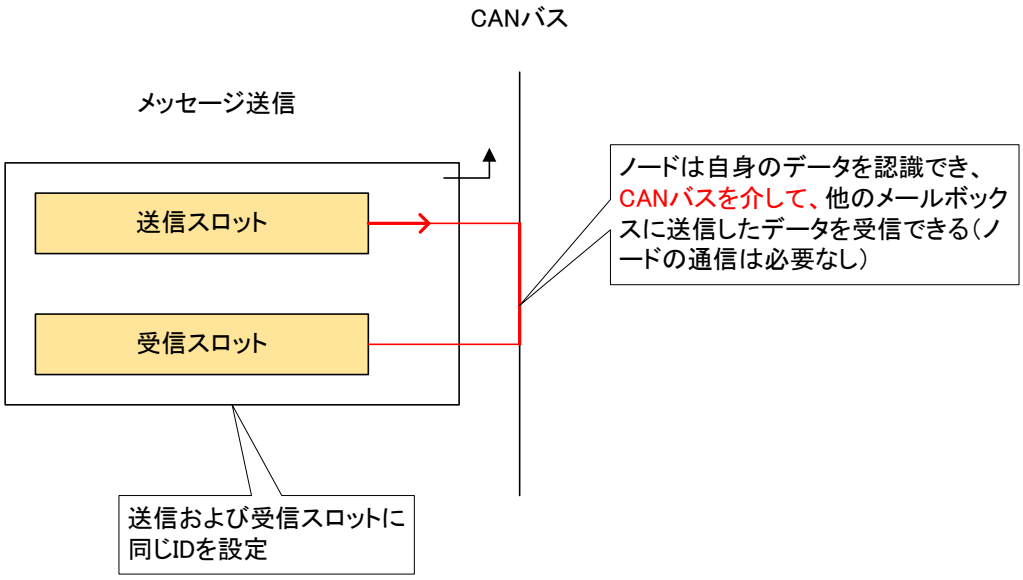

図 7　外部ループバック: CAN バスを介してメッセージを送信し、同じノードでメッセージを受信

これは、バス上の単一ノードでコードをテストするときに有用です。

5.2 リッスンオンリ（バスモニタ）

リッスンオンリモード、いわゆるバスモニタモードでは、ノードは ACK やエラーフレームなどを送信しません。この方法では、バストラフィックに影響することなくノードをテストできます。

【注意】

1. リッスンオンリモードのノードからフレームを送信しないでください。これは不正な動作であり、CAN モジュールでは対応していません。
2. ネットワークにあるノードが 2 つのみで、そのうち 1 つがリッスンオンリモードだった場合、他方のノードは ACK を受信せずに、送信を繰り返します。
3. リッスンオンリモードへの遷移箇所をコード内で明確にし、再度リッスンオンリモードを無効にすることを忘れないようにしてください。

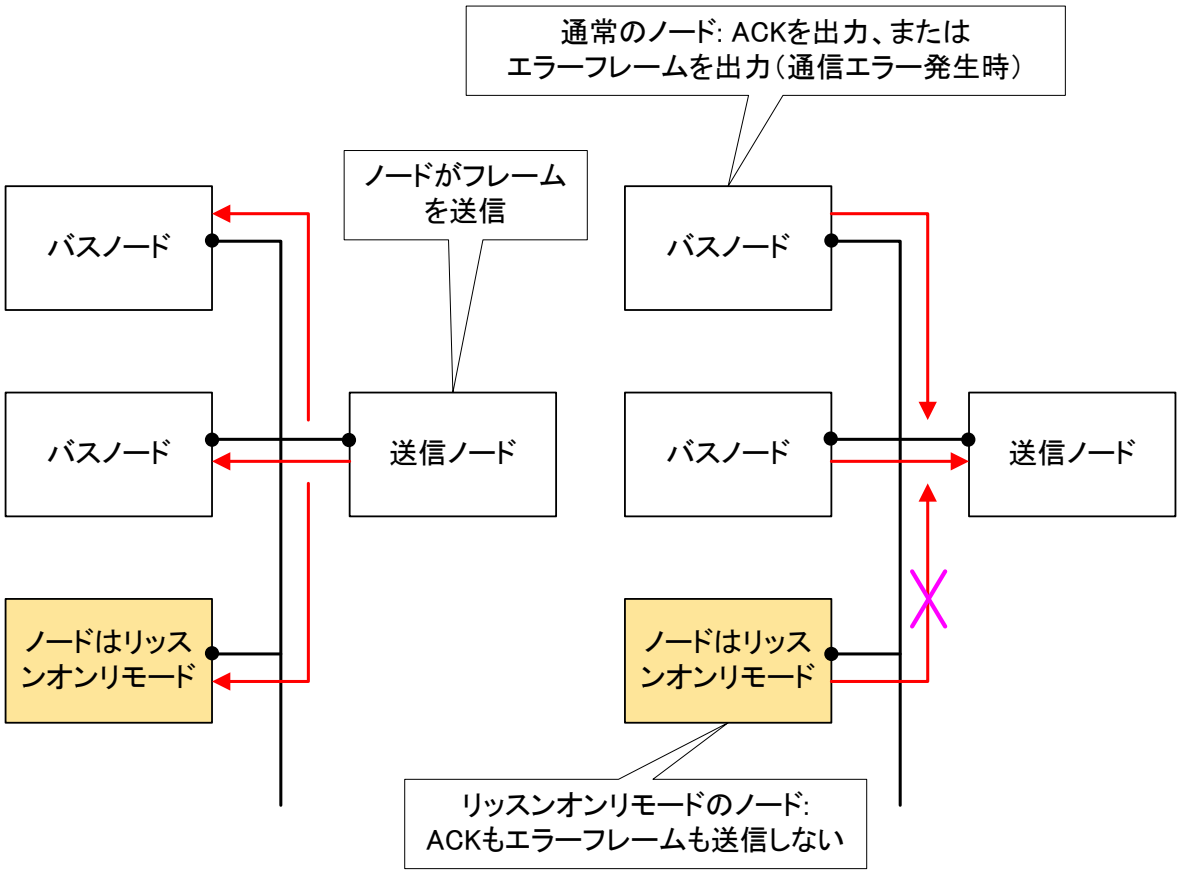

図 8 リッスンオンリモードのノード: ACK もエラーメッセージも送信しない

既存の CAN バスに新規のノードを追加する場合に、リッスンオンリモードは有用です。新たに接続されたノードを実際に稼働する前に、そのノードがフレームを正しく受信できるかどうかを確認できます。

これは新規ユニットを実稼働する前にバスの通信速度を検出するための一般的な用法です。リッスンオンリモードは Bosch CAN 仕様ではありませんが、ビットレート検出においては、ISO-11898 に準拠することが要求されます。

6. タイムスタンプ

タイムスタンプ関数は、受信メッセージがメールボックスに取り込まれたときのタイムスタンプカウンタ値を取得します。例えば、タイムスタンプを確認することによって、複数の受信メールボックスにメッセージが散在している場合、メッセージの順序を判断できます。タイムスタンプの読み出しは API では行いませんので、メールボックスをポーリングして、戻り値が R_CAN_OK（メッセージ待機中）になったときに、メールボックスからタイムスタンプを読み出してください。

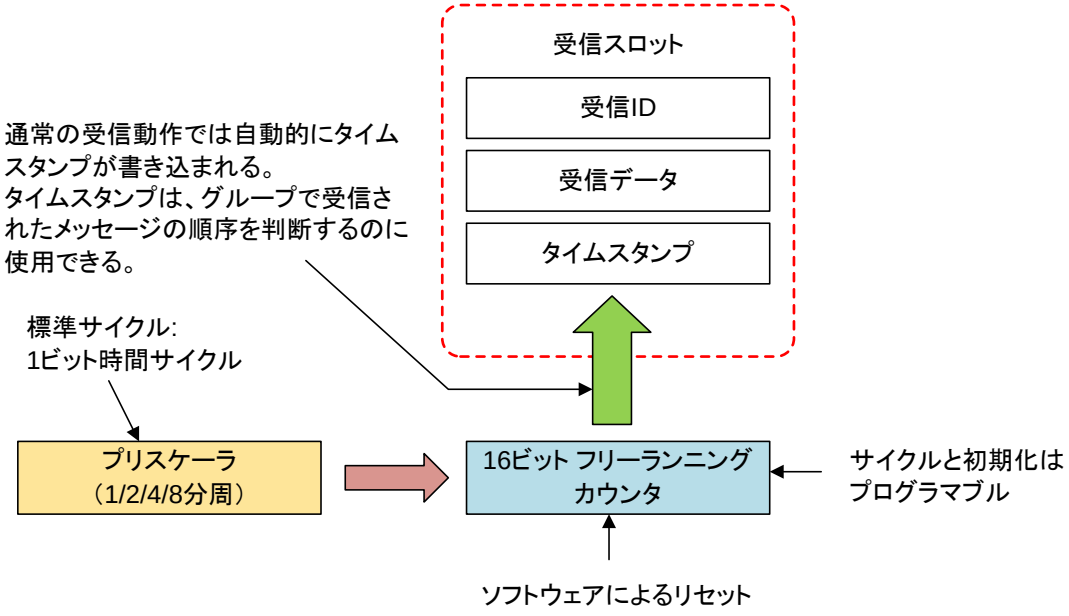

図 9 CAN のタイムスタンプ: 各メールボックスのタイムスタンプを使用可

7. CAN スリープモード

MCU リセット後の CAN のデフォルトモードは CAN スリープモードです。他の動作モードへは API を使って切り替えます（R_CAN_Control API 参照）。CAN スリープモードに遷移すると、CAN モジュールへのクロック供給が即座に停止され、消費電力を低減できます。CAN スリープモードへの遷移時、すべてのレジスタの状態はそのまま維持されます。

8. CAN FIFO

RX MCU では、CAN FIFO を使用できます。24 個のメールボックスが送信、または受信に設定されます。FIFO はポーリング、または割り込みと使用できます。

9. 付録

9.1 動作確認環境

このセクションでは、CAN 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 9.1 動作確認環境 (Rev.3.10)

項目	内容
統合開発環境（IDE）	ルネサスエレクトロニクス製 e^2 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 -WI,--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.3.10
使用ボード	Renesas Starter Kit+ for RX72M（型名：RTK5572Mxxxxxxxxxx）

表 9.2 動作確認環境 (Rev.3.00)

項目	内容
統合開発環境（IDE）	ルネサスエレクトロニクス製 e² studio V.7.3.0 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 -WI,--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.3.00
使用ボード	Renesas Starter Kit+ for RX65N-2M（型名 : RTK50565Nxxxxxxxxx）

9.2 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「This MCU is not supported by the current r_can_rx module.」エラーが発生します。

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

(3) 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「コンフィグ設定が間違っている場合のエラーメッセージ」エラーが発生します。

A：“r_can_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_can_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。

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。

TN-RX*-A151A/E

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
2.11	May.23.17	—	初版発行
		ﾌﾟﾛｸﾞﾗﾑ	RX64M および RX71M ユーザーズマニュアル ハードウェア 43.2.8 章の注記（SENTDATA ビットと TRMREQ ビットは同時に“0”に設定できない）に関連して、r_can_rx.c で R_CAN_TxCheck()および R_CAN_TxStopMsg()関数を変更。 MIXED_ID_MODE を使用する場合のみ、要求されたフレームタイプに応じて IDE ビットを設定するように変更。 R_CAN_RxRead()関数内で MIXED_ID_MODE モードを変更。 R_CAN_Control()関数内のケースで、EXITSLEEP_CANMODE、ENTERSLEEP_CANMODE、OPERATE_CANMODE の処理を変更。 ユーザが開発およびテスト時に、バスの問題を診断できるように、can_api_demo.c にユーザレベルの CAN エラー診断を追加。このコードはマクロで、ERROR_DIAG を“1”に設定することで有効になります。 USE_LCD コードをすべて削除。代わってデバッグコンソール（printf）を使用。デモにこれに対応したトレースコードを追加。 レガシー目的の API 以外の関数名を変更。 関数“handle_can_bus_state()” のコードを整理。
2.13	Jul.17.18	1	対象デバイスに RX66T を追加。
		9	R_CAN_Create()の説明を変更。R_CAN_RxSetMask ()および R_CAN_PortSet ()呼び出しへの参照を削除。
		24	R_CAN_RxSetMask()のコメントセクションのテキストを修正。 CAN_ERS_ISR()でチャネル 1 と 2 の ICU.GRPBE0.BIT を変更（すべてをチャネル 0 に設定）。
		28	図 4 のテキストを「TEC または REC > 127」から「TEC < 128 かつ REC <128」へ変更。
		31	7.2 リモートフレーム用に USE_CAN_POLL の値を"0"に修正。
2.14	Nov.16.18	全体	ドキュメントの全体的な構成や表現を見直し
2.15	Jan.10.19	1	対象デバイスに RX72T を追加。
3.00	May.20.19	—	以下のコンパイラをサポート。 - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		7	「2.3 ソフトウェアの要求」r_bsp v5.20 以上が必要
		12	「2.9 コードサイズ」セクションを更新。
		46	表 9.1「動作確認環境 (Rev. 3.00)」：更新。
		47	「9.2 トラブルシューティング」のセクションを追加。
		48	「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. 「evenaccess」を、BSP のマクロ定義で置き換えた。 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。 RTOS を使用している場合や、複数の割り込みを有効にしている場合に発生する、複数の周辺装置機能の間でのレジスタアクセスの競合（register access contention）を防止するために、処理に変更を加えた。 1. IEN (Interrupt Request Enable、割り込み要求の有効化) ビットのセット処理（setting process）を変更。 「Description」 BSP の API 関数内で R_BSP_InterruptRequestDisable と R_BSP_InterruptRequestEnable を使用するように、IEN (割り込み要求の有効化) ビットのセットプロセス（setting process）を変更。

			2. GENBL1 (Group Interrupt Request Enable Register、グループの割り込み要求の有効化レジスタ) ビットのセット処理を変更 (RX64M、RX65N、RX66T、RX71M、および RX72T)。 「Description」 割り込みが無効になっている間に、GENBL1 (グループの割り込み要求の有効化レジスタ) ビットのセット処理を実行するように変更。
3.10	Aug.15.19	1 12 46 プログラム	RX72M のサポートを追加。 RX72M に対応するコードサイズを追加。 「6.1 動作確認環境」： Rev.3.10 に対応する表を追加。 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 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

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

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

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

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

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/