RL78 ファミリ

MIDI インタフェースモジュール Software Integration System

要旨

本アプリケーションノートは、Software Integration System (SIS) を使用した MIDI インタフェースモジュールについて説明します。MIDI インタフェースモジュールは、ルネサス エレクトロニクス製 RL78 ファミリ MCU 内蔵のシリアル・アレイ・ユニット (SAU) の UART 機能、又はシリアル・インタフェース UARTA を使用して、MIDI 機器との通信制御を行います。以降、MIDI インタフェースモジュールを”本モジュール”と称します。

本モジュールの位置づけを以下に示します。

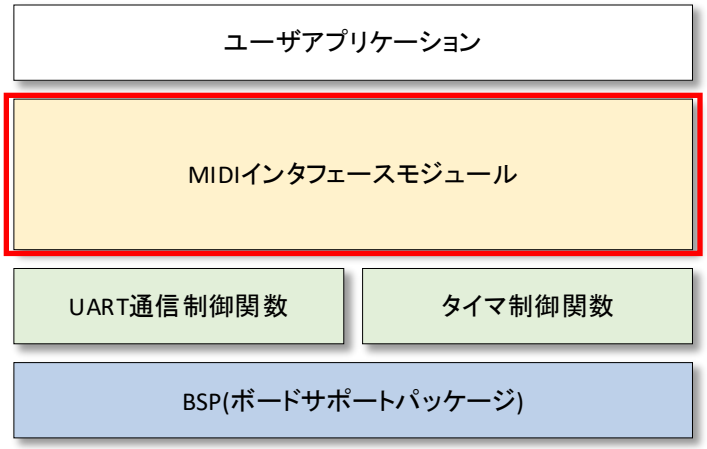


図 1 本モジュールの位置づけ

本モジュールは MIDI 1.0 規格に準拠しています。

詳細は音楽電子事業協会のサイトをご確認ください。

https://amei.or.jp/

備考. MIDI は、一般社団法人音楽電子事業協会 (AMEI) の登録商標です。

対象デバイス

・RL78 ファミリ

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。

関連ドキュメント

下記の [1] と [2] は本モジュールをお試しいただけるサンプルソフトウェアです。

- [1] RL78 ファミリ SIS を用いた MIDI 連動イルミネーション制御サンプルソフトウェア (R01AN7463)
- [2] RL78 ファミリ SIS を用いた MIDI 演奏制御サンプルソフトウェア (R01AN7491)
- [3] RL78 ファミリ ボードサポートパッケージモジュール Software Integration System (R01AN5522)
- [4] スマート・コンフィグレータ ユーザーズマニュアル RL78 API リファレンス編 (R20UT4852)
- [5] RL78 スマート・コンフィグレータ ユーザーガイド e2 studio 編 (R20AN0579)
- [6] RL78 スマート・コンフィグレータ ユーザーガイド IAR 編 (R20AN0581)
- [7] RL78/G16 Fast Prototyping Board ユーザーズマニュアル (R12UM0048)
- [8] 音楽電子事業協会 (リンク)
- [9] Arduino MIDI Library (リンク)

目次

1. 概要 5
1.1 MIDI インタフェースモジュールとは 5
1.2 MIDI インタフェースモジュールの概要 7
1.3 ハードウェア設定 8
1.4 API の概要 9
1.5 状態遷移図 10

2. API 情報 11
2.1 ハードウェアの要求 11
2.2 ソフトウェアの要求 11
2.3 サポートされているツールチェーン 11
2.4 使用する割り込みベクタ 11
2.5 ヘッダファイル 11
2.6 整数型 11
2.7 コンパイル時の設定 12
2.8 コードサイズ 14
2.9 引数 15
2.10 戻り値 19
2.11 SIS モジュールの追加方法 20

3. API 関数 21
3.1 R_MIDI_Open() 21
3.2 R_MIDI_Close() 22
3.3 R_MIDI_Begin() 23
3.4 R_MIDI_SendNoteOn() 24
3.5 R_MIDI_SendNoteOff() 25
3.6 R_MIDI_SendProgramChange() 26
3.7 R_MIDI_SendControlChange() 27
3.8 R_MIDI_Send() 28
3.9 R_MIDI_Read() 30
3.10 R_MIDI_GetType() 32
3.11 R_MIDI_GetData1() 33
3.12 R_MIDI_GetData2() 34
3.13 R_MIDI_GetChannel() 35
3.14 R_MIDI_SetInputChannel() 36
3.15 R_MIDI_SetThruFilterMode() 37
3.16 R_MIDI_NotifyEvent() 38
3.17 R_MIDI_Notify1msCycle() 39

4. 本モジュール追加後の各種設定 40
4.1 UART 通信機能の追加 40
4.1.1 UART コンポーネントの追加 40
4.1.2 通信設定 42
4.1.3 端子設定 45
4.1.4 ソースコードの追加 46

4.1.5　MIDI インタフェースモジュールとの紐づけ 46
4.1.6　初期設定処理 46
4.2　タイマ機能の追加 47
4.2.1　タイマコンポーネントの追加 47
4.2.2　タイマ設定 48
4.2.3　タイマ割り込みでのソースコードの追加 48
4.2.4　main プログラムでのソースコードの追加 49
4.2.5　初期設定処理 49

5.　参考ドキュメント 50

1. 概要

1.1 MIDI インタフェースモジュールとは

本モジュールは Arduino MIDI Library をベースに SIS モジュールとして組み込み可能にした MIDI 制御ソフトウェアです。

MIDI IN 端子からの MIDI メッセージ入力、および MIDI OUT 端子への MIDI メッセージ出力（2 端子をまとめて”MIDI インタフェース”と称します）を実現します。

本モジュールはスマート・コンフィグレータのコード生成などルネサスが無償提供しているソフトウェアモジュール（図 1-1、表 1-1 参照）と組み合わせて使用することにより MIDI 対応機器との制御が可能になります。

本モジュールは API として、プロジェクトに組み込んで使用します。本モジュールの組み込み方については、「2.11 SIS モジュールの追加方法」を参照してください。

本モジュールを使用して MIDI 通信を行う場合のソフトウェア構成を図 1-1 に示します。

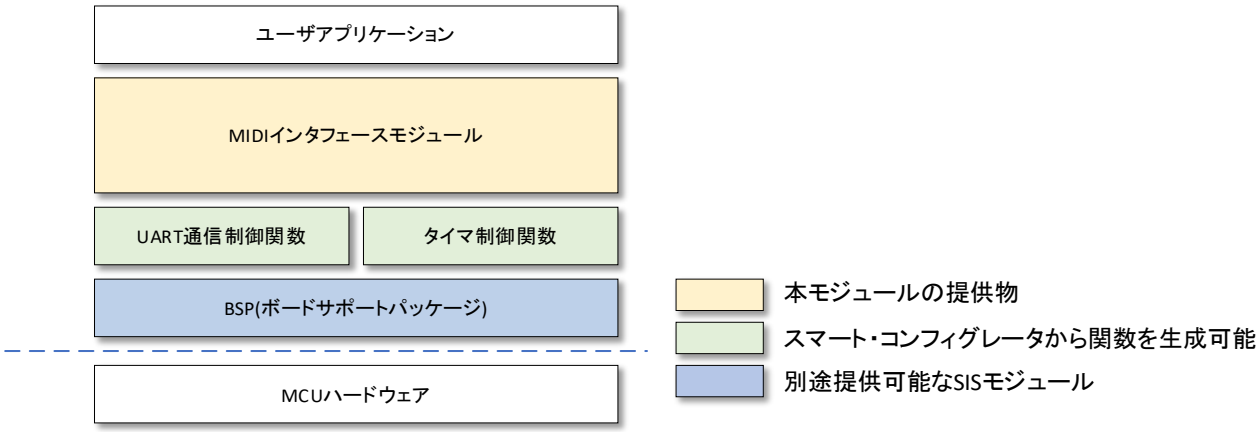


図 1-1　ソフトウェア構成

表 1-1 にアプリケーション構成で使用するモジュール一覧を示します。

表 1-1　アプリケーション構成で使用するモジュール一覧

名称	コンポーネント名称	コンポーネントタイプ
BSP (ボードサポートパッケージモジュール)	Board Support Package	RL78 Software Integration System
UART 通信制御	UART	コード生成
タイマ制御	インターバル・タイマ	コード生成

(1)　ボードサポートパッケージモジュール(BSP)

クロックの設定など、初期設定を行います。設定はスマート・コンフィグレータ上から行います。

(2)　UART 通信制御

外部接続した MIDI 機器と UART による MIDI 通信を行うためのコンポーネントです。設定はスマート・コンフィグレータ上から行います。

(3)　タイマ制御

本モジュール内部で経過時間を管理するために使用します。本モジュールのコンフィグレーションの設定内容により、使用が必要となります。設定はスマート・コンフィグレータ上から行います。

1.2　MIDI インタフェースモジュールの概要

本モジュールは UART を使用して MIDI 機器との通信制御を行います。

表 1-2 に本モジュールの機能を示します。

表 1-2　MIDI インタフェースモジュール機能一覧

項目	機能
ベースのソフトウェア	Arduino MIDI Library v5.0.2
準拠している規格	MIDI 規格 1.0
MIDI インタフェース制御可能数	1 つまで可能 (コンフィグレーションで設定)
アクティブセンシング機能	サポート (コンフィグレーションで設定)
スルー機能	ソフトウェアによるスルー出力をサポート (API 関数呼び出しで設定変更可能)
受信メッセージ通知	メッセージに応じたコールバック関数登録、関数呼び出し機能をサポート
MIDI チャンネル単位の受信フィルタリング	・すべてのチャンネルを受信可能 ・指定の 1 チャンネルのみ受信可能 ・すべてのチャンネルを受信抑止 のいずれかを指定可能 (API 関数呼び出し設定変更可能)
スタンダード MIDI ファイル規格対応	非サポート

1.3　ハードウェア設定

図 1-2 にハードウェア構成例を示します。

外部機器とは市販の MIDI ケーブルで接続します。

MCU の端子名は使用するシリアル・インタフェースにより異なります。使用するボードの端子と機能を参照し、使用する MCU の端子に読み替えてください。

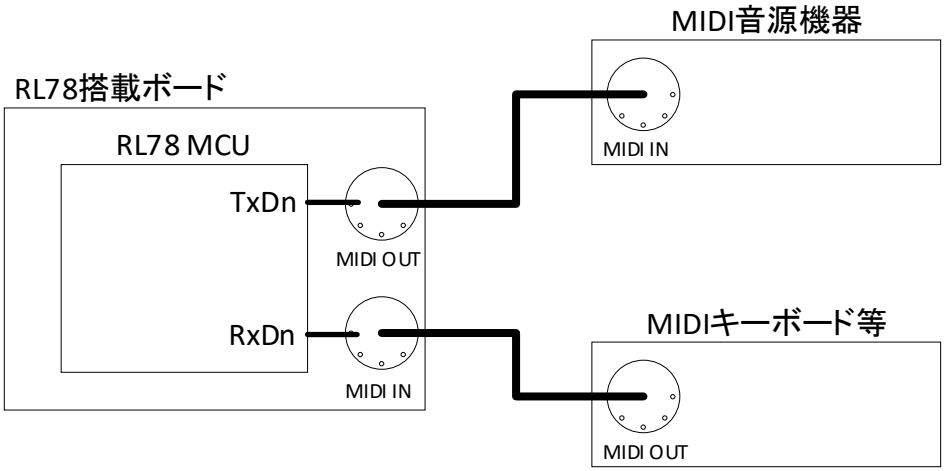


図 1-2 ハードウェア構成例

表 1-3 使用端子と機能

端子名	入出力	内容
TxDn	出力	UART データ送信
RxDn	入力	UART データ受信

1.4　API の概要

表 1-4 に本モジュールに含まれる API 関数を示します。

表 1-4　API 関数一覧

関数	関数説明
R_MIDI_Open()	モジュールのオープン処理
R_MIDI_Close()	モジュールのクローズ処理
R_MIDI_Begin()	モジュールの通信開始処理
R_MIDI_SendNoteOn()	NoteOn メッセージ送信処理
R_MIDI_SendNoteOff()	NoteOff メッセージ送信処理
R_MIDI_SendProgramChange()	ProgramChange メッセージ送信処理
R_MIDI_SendControlChange()	ControlChange メッセージ送信処理
R_MIDI_Send()	メッセージ送信共通処理
R_MIDI_Read()	メッセージ受信処理
R_MIDI_GetType()	受信したメッセージの種別情報取得処理
R_MIDI_GetChannel()	受信したメッセージのチャンネル情報取得処理
R_MIDI_GetData1()	受信したメッセージのデータ 1 取得処理
R_MIDI_GetData2()	受信したメッセージのデータ 2 取得処理
R_MIDI_SetInputChannel()	受信対象チャンネル設定処理
R_MIDI_SetThruFilterMode()	スルーモード設定処理
R_MIDI_NotifyEvent()	UART 通信の送信完了および受信通知コールバック処理
R_MIDI_Notify1msInterval() 注 1	インターバルタイマカウンタ処理

注 1 : アクティブセンシング機能を有効にする場合、時間管理が必要になるため、ハードウェアタイマやソフトウェアタイマ等を使って、1ms 間隔でコールする必要があります。

1.5　状態遷移図

図 1-3 に本モジュールの状態遷移図を示します。

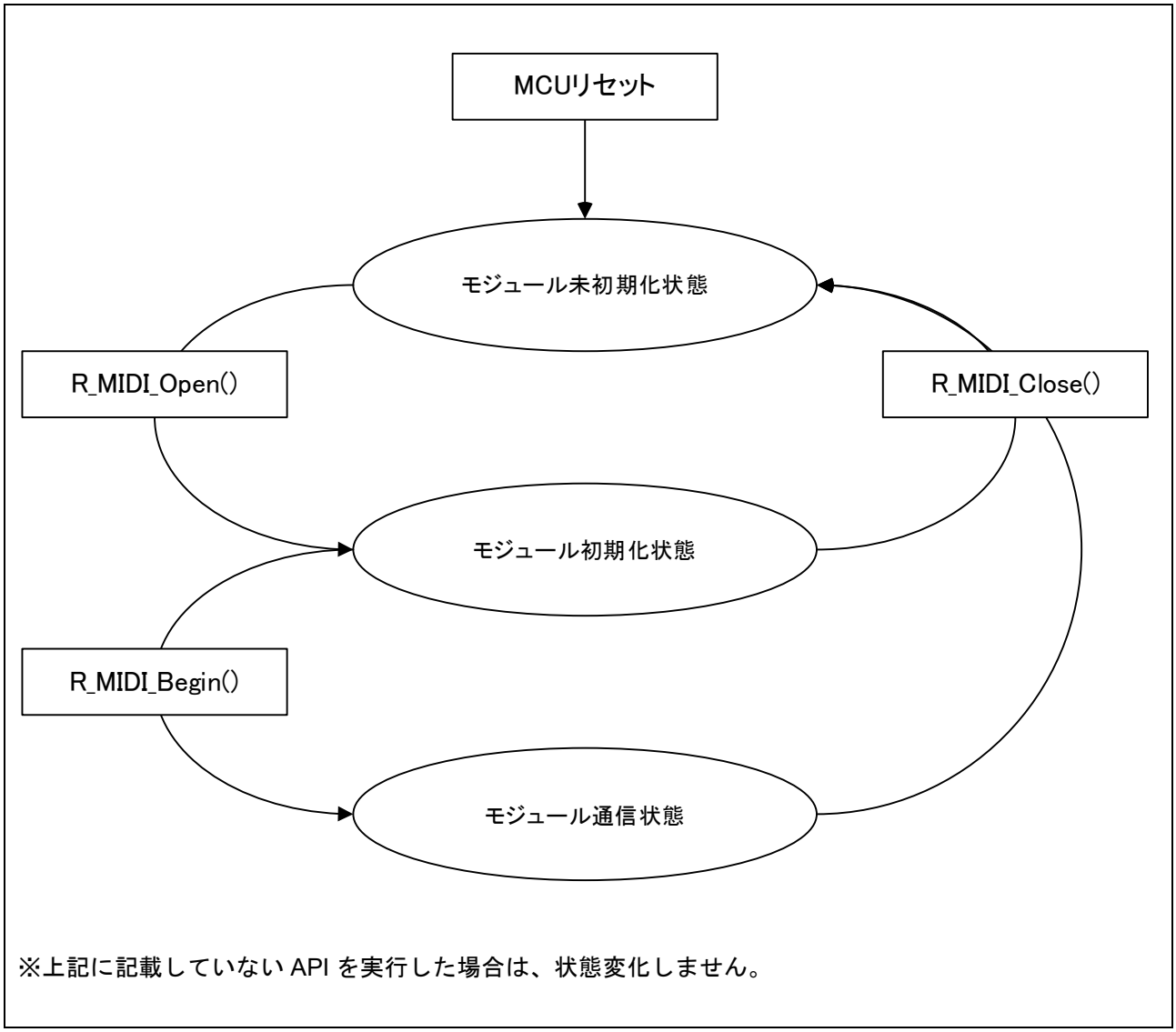


図 1-3　MIDI インタフェースモジュールの状態遷移図

2. API 情報

本モジュールは、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 が以下のいずれかの機能をサポートしている必要があります。

- CSI(UART)
- UARTA

2.2 ソフトウェアの要求

このモジュールは以下の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) Rev.1.62 以降

また、以下のコード生成に依存しています。

- UART

2.3 サポートされているツールチェーン

本モジュールは以下に示すツールチェーンで動作確認を行っています。

- Renesas CC-RL Toolchain v1.13.00
- IAR Embedded Workbench for Renesas RL78 v5.10.3

2.4 使用する割り込みベクタ

なし

2.5 ヘッダファイル

すべての API コールとそれをサポートするインタフェースの定義は以下の通りです。

r_midi_rl78_if.h　：ユーザが本モジュールを用いて実装する場合にインクルードするファイルです。
r_midi_rl78_api.h　：API の定義を記載しています。
r_midi_rl78_config.h　：コンフィグレーションオプションの設定ファイルです。

2.6 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。

2.7　コンパイル時の設定

本モジュールのコンフィグレーションオプションの設定は、r_midi_rl78_config.h で行います。

スマート・コンフィグレータ を使用する場合は、ソフトウェアコンポーネント設定画面でコンフィグレーションオプションを設定できます。設定値はモジュールを追加する際に、自動的に r_midi_rl78_config.h に反映されます。オプション名および設定値に関する説明を、下表に示します。

Configuration options in r_midi_rl78_config.h	
MIDI_CFG_PARAM_CHECKING_ENABLE ※デフォルト値は”BSP_CFG_PARAM_CHECKING_ENABLE”	1：ビルド時にパラメータチェックの処理をコードに含めます。 0：ビルド時にパラメータチェックの処理をコードから省略します。 このオプションに BSP_CFG_PARAM_CHECKING_ENABLE を設定すると、システムのデフォルト設定が使用されます。
MIDI_CFG_CTRL_NUM_MAX ※デフォルト値は”1”	MIDI インタフェースの個数を定義します。 ※本バージョンでは 1 のみ指定可能です。
MIDI_CFG_C0_USE_RUNNING_STATUS ※デフォルト値は”false”	連続して同じステータスメッセージを送信する際に、ステータス・バイトの送信を省略するかどうかを定義します。 true：省略します。 false：省略しません。
MIDI_CFG_C0_HNDL_NULL_VELOCITY ※デフォルト値は”true”	velocity = 0 の NoteOn メッセージを受信したときのユーザへの通知方法を定義します。 true：NoteOff メッセージとして扱います。 false：NoteOn メッセージとして扱います。
MIDI_CFG_C0_USE_1BYTE_PARSING ※デフォルト値は”true”	R_MIDI_Read()実行時の受信データの解析手法について設定します。 true：R_MIDI_Read()を 1 回実行するごとに受信データを 1 バイトだけ解析して終了します。 false：R_MIDI_Read()を 1 回実行するごとに受信バッファに溜まっているデータをすべて解析します。
MIDI_CFG_C0_SYSEX_MAX_SIZE ※デフォルト値は”128”	SystemExclusive メッセージを格納するバッファのサイズを定義します。 4~1024 の範囲で偶数値を指定してください。
MIDI_CFG_C0_USE_SND_ACTSENSE ※デフォルト値は”false”	MIDI OUT 端子に定期的にアクティブセンシングメッセージを送信するかどうかを設定します。 true：送信します。 false：送信しません。
MIDI_CFG_C0_ USE_REV_ACTSENSE ※デフォルト値は”false”	MIDI IN 端子に接続した機器から受信するアクティブセンシングメッセージを監視するかどうかを設定します。 true：監視します。 false：監視しません。
MIDI_CFG_C0_SND_ACTSENSE_PERIOD ※デフォルト値は”0”	アクティブセンシングメッセージの送信間隔を設定します。 0~250 の範囲で設定してください。 0 を設定した場合：送信しません。
MIDI_CFG_C0_UART_COMPONENT ※デフォルト値は”Config_UART0”	MIDI 通信を行う UART 通信のコンフィグレーション名を定義します。 この定義はスマート・コンフィグレータの UART 通信で設定した UART 通信の関数名生成に使用します。
MIDI_CFG_C0_UART_TXBUF_SIZE ※デフォルト値は”80”	MIDI 通信を行う際の送信用バッファサイズを定義します。 4~1024 の範囲で偶数値を設定して下さい。

Configuration options in r_midi_rl78_config.h	
MIDI_CFG_C0_UART_RXBUF_SIZE ※デフォルト値は”80”	MIDI 通信を行う際の受信用バッファサイズを定義します。 4~1024 の範囲で偶数値を設定して下さい。

2.8　コードサイズ

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

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

下表の値は下記条件で確認しています。

モジュールリビジョン: r_midi_rl78 rev.1.00

コンパイラバージョン: Renesas Electronics RL78 Family C Compiler Package V1.13.00

(統合開発環境のデフォルト設定)

IAR C/C++ Compiler for Renesas RL78 version 5.10.3

(統合開発環境のデフォルト設定)

コンフィグレーションオプション: デフォルト設定

ROM、RAM およびスタックのコードサイズ

デバイス	分類	使用メモリ					
		Renesas Compiler				IAR Compiler	
		最適化レベル -Olite		最適化レベル -Odefault		Optmization level (-Og) レベル 低	
		パラメータチェック		パラメータチェック		パラメータチェック	
		あり	なし	あり	なし	あり	なし
RL78/G16	ROM	6619	6086	5986	5497	6755	5848
	RAM	461		461		419	
	最大使用スタックサイズ	90		88		86	

2.9　引数

API 関数の引数である構造体を示します。この構造体は、API 関数のプロトタイプ宣言とともに r_midi_rl78_api.h に記載されています。

(1)　コントロール構造体定義

MIDI インタフェース構成のコントロール構造体です。アプリケーションからの設定は不要です。

スマート・コンフィグレータからコード生成することで本モジュールの構成に応じた変数が出力されるので、このモジュールの API の第 1 引数にこの変数の p_ctrl メンバを指定してください。

```
typedef struct midi_instance_ctrl_tag
{
   /* Arduino MIDI Library Variables */
   void (*mMessageCallback)(MidiMessage *);
   ErrorCallback mErrorCallback;
   NoteOffCallback mNoteOffCallback;
   NoteOnCallback mNoteOnCallback;
   AfterTouchPolyCallback mAfterTouchPolyCallback;
   ControlChangeCallback mControlChangeCallback;
   ProgramChangeCallback mProgramChangeCallback;
   AfterTouchChannelCallback mAfterTouchChannelCallback;
   PitchBendCallback mPitchBendCallback;
   SystemExclusiveCallback mSystemExclusiveCallback;
   TimeCodeQuarterFrameCallback mTimeCodeQuarterFrameCallback;
   SongPositionCallback mSongPositionCallback;
   SongSelectCallback mSongSelectCallback;
   TuneRequestCallback mTuneRequestCallback;
   ClockCallback mClockCallback;
   StartCallback mStartCallback;
   TickCallback mTickCallback;
   ContinueCallback mContinueCallback;
   StopCallback mStopCallback;
   ActiveSensingCallback mActiveSensingCallback;
   SystemResetCallback mSystemResetCallback;
   Channel        mInputChannel;
   StatusByte     mRunningStatus_RX;
   StatusByte     mRunningStatus_TX;
   byte           mPendingMessage[3];
   unsigned short  mPendingMessageExpectedLength;
   unsigned short  mPendingMessageIndex;
   unsigned short  mCurrentRpnNumber;
   unsigned short  mCurrentNrpnNumber;
   bool           mThruActivated;
   midi_thru_mode_t      mThruFilterMode;
   MidiMessage    mMessage;
   unsigned long   mLastMessageSentTime;
   unsigned long   mLastMessageReceivedTime;
   unsigned long   mSenderActiveSensingPeriodicity;
   bool           mReceiverActiveSensingActivated;
   int8_t         mLastError;
   /* Arduino MIDI Library Variables */

   /* Arduino MIDI Library Configuration */
   midi_settings_t  *p_Settings;
   /* Arduino MIDI Library Configuration */

   /* Serial Control */
   midi_uart_ctrl_t  * p_uart_ctrl;
   uint32_t open;
   midi_cfg_t const       * p_cfg; ///< middleware configuration.
   midi_bus_extended_cfg_t * p_bus; ///< Bus using this device;
   /* Serial Control */
}midi_instance_ctrl_t;
```

(2)　コンフィグレーション構造体定義

スマート・コンフィグレータからコード生成することで本モジュールの構成に応じた変数が出力されるので、このモジュールの API の第 2 引数にこの変数の p_cfg メンバを指定してください。

```
typedef struct st_midi_cfg
{
   midi_settings_t * midi_settings;
   void const * p_extend;           ///< Pointer to extended configuration by instance of interface.
   uint8_t * p_sysex_array;
   uint16_t tx_ring_buff_size;
   uint16_t rx_ring_buff_size;
} midi_cfg_t;
```

上記構造体で使用している構造体を示します。

(3)　midi_settings_t 構造体定義

```
typedef struct _Settings_tag
{
   bool UseRunningStatus;
   bool HandleNullVelocityNoteOnAsNoteOff;
   bool Use1ByteParsing;
   uint16_t SysExMaxSize;
   bool UseSenderActiveSensing;
   bool UseReceiverActiveSensing;
   uint16_t SenderActiveSensingPeriodicity;
}midi_settings_t;  //_Settings
```

(4)　マクロ

本モジュールが使用するマクロ・列挙型を以下に示します。

・midi_type_t 型

```
typedef enum e_midi_type
{
      MIDI_TYPE_InvalidType         = (uint8_t)0x00,    ///< For notifying
errors
      MIDI_TYPE_NoteOff             = 0x80,    ///< Channel Message - Note Off
      MIDI_TYPE_NoteOn              = 0x90,    ///< Channel Message - Note On
      MIDI_TYPE_AfterTouchPoly       = 0xA0,    ///< Channel Message - Polyphonic
AfterTouch
      MIDI_TYPE_ControlChange        = 0xB0,    ///< Channel Message - Control
Change / Channel Mode
      MIDI_TYPE_ProgramChange        = 0xC0,    ///< Channel Message - Program
Change
      MIDI_TYPE_AfterTouchChannel    = 0xD0,    ///< Channel Message - Channel
(monophonic) AfterTouch
      MIDI_TYPE_PitchBend           = 0xE0,    ///< Channel Message - Pitch Bend
      MIDI_TYPE_SystemExclusive      = 0xF0,    ///< System Exclusive
      MIDI_TYPE_SystemExclusiveStart  = MIDI_TYPE_SystemExclusive,   ///< System
Exclusive Start
      MIDI_TYPE_TimeCodeQuarterFrame  = 0xF1,    ///< System Common - MIDI Time
Code Quarter Frame
      MIDI_TYPE_SongPosition         = 0xF2,    ///< System Common - Song
Position Pointer
      MIDI_TYPE_SongSelect          = 0xF3,    ///< System Common - Song Select
      MIDI_TYPE_Undefined_F4         = 0xF4,
      MIDI_TYPE_Undefined_F5         = 0xF5,
      MIDI_TYPE_TuneRequest          = 0xF6,    ///< System Common - Tune Request
      MIDI_TYPE_SystemExclusiveEnd    = 0xF7,    ///< System Exclusive End
      MIDI_TYPE_Clock               = 0xF8,    ///< System Real Time - Timing
Clock
      MIDI_TYPE_Undefined_F9         = 0xF9,
      MIDI_TYPE_Tick                = MIDI_TYPE_Undefined_F9, ///< System Real
Time - Timing Tick (1 tick = 10 milliseconds)
      MIDI_TYPE_Start               = 0xFA,    ///< System Real Time - Start
      MIDI_TYPE_Continue             = 0xFB,    ///< System Real Time - Continue
      MIDI_TYPE_Stop                = 0xFC,    ///< System Real Time - Stop
      MIDI_TYPE_Undefined_FD         = 0xFD,
      MIDI_TYPE_ActiveSensing        = 0xFE,    ///< System Real Time - Active
Sensing
      MIDI_TYPE_SystemReset          = 0xFF,    ///< System Real Time - System
Reset
} midi_type_t;
```

・midi_ccn_t 型

```
typedef enum e_midi_ccn
{
    // High resolution Continuous Controllers MSB (+32 for LSB) ----------------
    MIDI_CCN_BankSelect               = (uint8_t)0,
    MIDI_CCN_ModulationWheel           = 1,
    MIDI_CCN_BreathController          = 2,
    // CC3 undefined
    MIDI_CCN_FootController            = 4,
    MIDI_CCN_PortamentoTime            = 5,
    MIDI_CCN_DataEntryMSB              = 6,
    MIDI_CCN_ChannelVolume             = 7,
    MIDI_CCN_Balance                  = 8,
    // CC9 undefined
    MIDI_CCN_Pan                      = 10,
    MIDI_CCN_ExpressionController       = 11,
    MIDI_CCN_EffectControl1            = 12,
    MIDI_CCN_EffectControl2            = 13,
    // CC14 undefined
    // CC15 undefined
    MIDI_CCN_GeneralPurposeController1   = 16,
    MIDI_CCN_GeneralPurposeController2   = 17,
    MIDI_CCN_GeneralPurposeController3   = 18,
    MIDI_CCN_GeneralPurposeController4   = 19,

    MIDI_CCN_DataEntryLSB              = 38,

    // Switches ----------------------------------------------------------------
    MIDI_CCN_Sustain                  = 64,
    MIDI_CCN_Portamento                = 65,
    MIDI_CCN_Sostenuto                 = 66,
    MIDI_CCN_SoftPedal                 = 67,
    MIDI_CCN_Legato                   = 68,
    MIDI_CCN_Hold                     = 69,

    // Low resolution continuous controllers -----------------------------------
    MIDI_CCN_CoundController1          = 70,   ///< Synth: Sound Variation   FX: Exciter On/Off
    MIDI_CCN_SoundController2          = 71,   ///< Synth: Harmonic Content  FX: Compressor On/Off
    MIDI_CCN_SoundController3          = 72,   ///< Synth: Release Time     FX: Distortion On/Off
    MIDI_CCN_SoundController4          = 73,   ///< Synth: Attack Time      FX: EQ On/Off
    MIDI_CCN_SoundController5          = 74,   ///< Synth: Brightness       FX: Expander On/Off
    MIDI_CCN_SoundController6          = 75,   ///< Synth: Decay Time       FX: Reverb On/Off
    MIDI_CCN_SoundController7          = 76,   ///< Synth: Vibrato Rate     FX: Delay On/Off
    MIDI_CCN_SoundController8          = 77,   ///< Synth: Vibrato Depth    FX: Pitch Transpose
On/Off
    MIDI_CCN_SoundController9          = 78,   ///< Synth: Vibrato Delay    FX: Flange/Chorus On/Off
    MIDI_CCN_SoundController10         = 79,   ///< Synth: Undefined        FX: Special Effects
On/Off
    MIDI_CCN_GeneralPurposeController5   = 80,
    MIDI_CCN_GeneralPurposeController6   = 81,
    MIDI_CCN_GeneralPurposeController7   = 82,
    MIDI_CCN_GeneralPurposeController8   = 83,
    MIDI_CCN_PortamentoControl         = 84,
    // CC85 to CC90 undefined
    MIDI_CCN_Effects1                 = 91,   ///< Reverb send level
    MIDI_CCN_Effects2                 = 92,   ///< Tremolo depth
    MIDI_CCN_Effects3                 = 93,   ///< Chorus send level
    MIDI_CCN_Effects4                 = 94,   ///< Celeste depth
    MIDI_CCN_Effects5                 = 95,   ///< Phaser depth
    MIDI_CCN_DataIncrement             = 96,
    MIDI_CCN_DataDecrement             = 97,
    MIDI_CCN_NRPNLSB                  = 98,   ///< Non-Registered Parameter Number (LSB)
    MIDI_CCN_NRPNMSB                  = 99,   ///< Non-Registered Parameter Number (MSB)
    MIDI_CCN_RPNLSB                   = 100,  ///< Registered Parameter Number (LSB)
    MIDI_CCN_RPNMSB                   = 101,  ///< Registered Parameter Number (MSB)

    // Channel Mode messages ---------------------------------------------------
    MIDI_CCN_AllSoundOff               = 120,
    MIDI_CCN_ResetAllControllers        = 121,
    MIDI_CCN_LocalControl              = 122,
    MIDI_CCN_AllNotesOff               = 123,
    MIDI_CCN_OmniModeOff               = 124,
    MIDI_CCN_OmniModeOn                = 125,
    MIDI_CCN_MonoModeOn                = 126,
    MIDI_CCN_PolyModeOn                = 127
} midi_ccn_t;
```

・midi_thru_mode_t 型

```
typedef enum e_midi_thru_mode
{
    MIDI_THRU_Off               = 0,  ///< Thru disabled (nothing passes through).
    MIDI_THRU_Full               = 1,  ///< Fully enabled Thru (every incoming message is sent back).
    MIDI_THRU_SameChannel        = 2,  ///< Only the messages on the Input Channel will be sent back.
    MIDI_THRU_DifferentChannel   = 3,  ///< All the messages but the ones on the Input Channel will
be sent back.
} midi_thru_mode_t;
```

2.10　戻り値

API 関数の戻り値を示します。この列挙型は、ボードサポートパッケージの fsp_common_api.h で記載されています。

```
Input Channel will be sent back.
/** Common error codes */
typedef enum e_fsp_err
{
   FSP_SUCCESS = 0,

   FSP_ERR_ASSERTION          = 1,    ///< A critical assertion has failed
   FSP_ERR_INVALID_ARGUMENT   = 3,    ///< Invalid input parameter
   FSP_ERR_NOT_OPEN           = 7,    ///< Requested channel is not configured or API not open
   FSP_ERR_ALREADY_OPEN       = 14,   ///< Requested channel is already open in a different
configuration
} fsp_err_t;
```

2.11　SIS 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。

(1) e² studio 上でスマート・コンフィグレータを使用して SIS モジュールを追加する場合
e² studio のスマート・コンフィグレータを使用して、自動的にユーザプロジェクトに SIS モジュールを追加します。詳細は、関連ドキュメント[3] を参照してください。

(2) IAREW 上でスマート・コンフィグレータを使用して SIS モジュールを追加する場合
スタンドアロン版スマート・コンフィグレータを使用して、自動的にユーザプロジェクトに SIS モジュールを追加します。詳細は、関連ドキュメント[4] を参照してください。

3. API 関数

3.1 R_MIDI_Open()

この関数は、本モジュールを初期化します。この関数は他の API 関数を使用する前に実行する必要があります。

Format

```
fsp_err_t R_MIDI_Open (
    midi_ctrl_t * const          p_api_ctrl,
    midi_cfg_t const * const     p_cfg
)
```

Parameters

p_api_ctrl

コントロール構造体へのポインタ

p_cfg

コンフィグレーション構造体のポインタ

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_ALREADY_OPEN	Close()のコールなしに Open()がコールされました
FSP_ERR_INVALID_ARGUMENT	コンフィグレーションのパラメータが不正です

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

引数 p_api_ctrl に指定されたコントロール構造体領域の初期化、引数 p_cfg に指定されたワーク領域との紐づけを行います。

クローズ処理を終了させるまで、コントロール構造体とワーク領域を保持し、その内容をアプリケーションプログラムで変更しないでください。

Example

```
#include "r_midi_rl78_if.h"

if (FSP_SUCCESS != R_MIDI_Open(g_midi_c0_instance.p_ctrl,
g_midi_c0_instance.p_cfg))
{
      /* Error */
}
```

Special Notes

本関数実行前に、UART 機能の初期設定が必要です。「4.1 UART 通信機能の追加」を参照してください。

3.2 R_MIDI_Close()

この関数は、本モジュールをクローズします。

Format

```
fsp_err_t R_MIDI_Close (
    midi_ctrl_t * p_api_ctrl
)
```

Parameters

p_api_ctrl

コントロール構造体へのポインタ

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

本モジュールの処理を終了し、リソースを解放します。

R_MIDI_Open()で引数に指定したコントロール構造体やコンフィグで指定した各種ワーク領域は、本関数実行後は使用されません。他用途に使用できます。

Example

```
if (FSP_SUCCESS != R_MIDI_Close(g_midi_c0_instance.p_ctrl))
{
    /* Error */
}
```

Special Notes

なし

3.3　R_MIDI_Begin()

この関数は外部 MIDI 機器との通信を開始します。

Format

```
fsp_err_t R_MIDI_Begin (
    midi_ctrl_t * const   p_api_ctrl,
    uint8_t               channel
)
```

Parameters

p_api_ctrl

コントロール構造体へのポインタ

channel

受信対象とする MIDI チャンネル

MIDI_CHANNEL_OMNI : すべての MIDI チャンネルを入力対象にします
1 ~ 16 : 指定した MIDI チャンネルを入力対象にします
MIDI_CHANNEL_OFF : すべての MIDI チャンネルを入力禁止にします

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

接続された機器との MIDI 通信を開始します。

引数で指定した MIDI チャンネルデータを受信すると、R_MIDI_Read()関数で取得できます。

本関数実行後、R_MIDI_SetInputChannel()を実行して受信対象の MIDI チャンネルを変更することができます。

Example

```
if (FSP_SUCCESS != R_MIDI_Begin(g_midi_c0_instance.p_ctrl, MIDI_CHANNEL_OMNI))
{
    /* Error */
}
```

Special Notes

なし

3.4　R_MIDI_SendNoteOn()

NoteOn メッセージを送信します。

Format

```
fsp_err_t R_MIDI_SendNoteOn (
      midi_ctrl_t * const   p_api_ctrl,
      uint8_t               inNoteNumber,
      uint8_t               inVelocity,
      uint8_t               inChannel
)
```

Parameters

p_api_ctrl

　コントロール構造体へのポインタ

inNoteNumber

　ノート番号 (0 ～ 127)

inVelocity：

　ベロシティ(0 ～ 127)

inChannel：

　MIDI チャンネル (1 ～ 16)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

MIDI OUT 端子に接続されている機器に NoteOn メッセージを送信します。

R_MIDI_Begin()実行後に、本関数を実行してください。

Example

```
if (FSP_SUCCESS != R_MIDI_SendNoteOn(g_midi_c0_instance.p_ctrl, 48, 127, 1))
{
      /* Error */
}
```

Special Notes

引数 inNoteNumber、inVelocity は下位 7 ビットの値が有効になります。

3.5　R_MIDI_SendNoteOff()

NoteOff メッセージを送信します。

Format

```
fsp_err_t R_MIDI_SendNoteOff (
      midi_ctrl_t * const   p_api_ctrl,
      uint8_t               inNoteNumber,
      uint8_t               inVelocity,
      uint8_t               inChannel
)
```

Parameters

p_api_ctrl

　コントロール構造体へのポインタ

inNoteNumber

　ノート番号 (0 ～ 127)

inVelocity：

　ベロシティ(0 ～ 127)

inChannel：

　MIDI チャンネル (1 ～ 16)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

MIDI OUT 端子に接続されている機器に NoteOff メッセージを送信します。

R_MIDI_Begin()実行後に、本関数を実行してください。

Example

```
if (FSP_SUCCESS != R_MIDI_SendNoteOff(g_midi_c0_instance.p_ctrl, 48, 127, 1))
{
      /* Error */
}
```

Special Notes

引数 inNoteNumber、inVelocity は下位 7 ビットの値が有効になります。

3.6　R_MIDI_SendProgramChange()

ProgramChange メッセージを送信します。

Format

```
fsp_err_t R_MIDI_SendProgramChange (
      midi_ctrl_t * const   p_api_ctrl,
      uint8_t               inProgramNumber,
      uint8_t               inChannel
)
```

Parameters

p_api_ctrl

　コントロール構造体へのポインタ

inProgramNumber

　プログラム番号（0 ～ 127）

inChannel：

　MIDI チャンネル（1 ～ 16）

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

MIDI OUT 端子に接続されている機器に ProgramChange メッセージを送信します。

R_MIDI_Begin()実行後に、本関数を実行してください。

Example

```
if (FSP_SUCCESS != R_MIDI_SendProgramChange(g_midi_c0_instance.p_ctrl, 0, 1))
{
      /* Error */
}
```

Special Notes

引数 inProgramNumber は下位 7 ビットの値が有効になります。

3.7　R_MIDI_SendControlChange()

ControlChange メッセージを送信します。

Format

```
fsp_err_t R_MIDI_SendControlChange (
    midi_ctrl_t * const     p_api_ctrl,
    midi_ccn_t              inControlNumber,
    uint8_t                 inControlValue,
    uint8_t                 inChannel
)
```

Parameters

p_api_ctrl

　コントロール構造体へのポインタ

inControlNumber

　コントロール番号（0 ～ 127）

inControlValue

　設定値（0 ～ 127）

inChannel：

　MIDI チャンネル（1 ～ 16）

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

MIDI OUT 端子に接続されている機器に ControlChange メッセージを送信します。

R_MIDI_Begin()実行後に、本関数を実行してください。

Example

```
if (FSP_SUCCESS != R_MIDI_SendControlChange(g_midi_c0_instance.p_ctrl,
MIDI_CCN_AllNotesOff, 0, 1))
{
      /* Error */
}
```

Special Notes

引数 inProgramNumber は下位 7 ビットの値が有効になります。

3.8　R_MIDI_Send()

MIDI 機器にメッセージを送信します。

Format

```
fsp_err_t R_MIDI_Send (
      midi_ctrl_t * const   p_api_ctrl,
      midi_type_t           inType,
      uint8_t               inData1,
      uint8_t               inData2,
      uint8_t               inChannel
)
```

Parameters

p_api_ctrl

　コントロール構造体へのポインタ

inType

　送信メッセージの種類

　指定可能なメッセージは MIDI_TYPE_NoteOff ～ MIDI_TYPE_PitchBend、

　または MIDI_TYPE_Clock ～ MIDI_TYPE_SystemReset

inData1

　設定値 1

inData2

　設定値 2

inChannel：

　MIDI チャンネル（1 ～ 16）

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

MIDI OUT 端子に接続されている機器にメッセージを送信します。

R_MIDI_Begin()実行後に、本関数を実行してください。

Example

```
if (FSP_SUCCESS != R_MIDI_Send(g_midi_c0_instance.p_ctrl, MIDI_TYPE_SystemReset,
0, 0, 0))
{
      /* Error */
}
```

Special Notes

inType に MIDI_TYPE_Clock ～ MIDI_TYPE_SystemReset を指定する場合、引数 inData1、inData2、inChannel には 0 を指定してください。

3.9　R_MIDI_Read()

MIDI 機器からの受信メッセージ通知を受け取ります。

Format

```
fsp_err_t R_MIDI_Read (
    midi_ctrl_t * const   p_api_ctrl,
    bool *                p_result
)
```

Parameters

p_api_ctrl

コントロール構造体へのポインタ

*p_result

受信メッセージ通知の結果を格納する変数のポインタ

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

MIDI IN 端子に接続されている機器からメッセージ受信があるかどうかを確認し、その結果を p_result に指定された領域に格納します。

結果の値が true の場合は受信あり、false の場合は受信なし、を意味します。

Example

```
bool        result = false;
midi_type_t type;
Channel     channel;
uint8_t     data1;
uint8_t     data2;

if (FSP_SUCCESS != R_MIDI_Read(g_midi_c0_instance.p_ctrl, &result))
{
      /* Error */
}
else
{
      if (true == result)
      {
          /* Receipt notification available */
          R_MIDI_GetType(g_midi_c0_instance.p_ctrl, &type);
          R_MIDI_GetData1(g_midi_c0_instance.p_ctrl, &data1);
          R_MIDI_GetData2(g_midi_c0_instance.p_ctrl, &data2);
          R_MIDI_GetChannel(g_midi_c0_instance.p_ctrl, &channel);
      }

}
```

Special Notes

スルーモードが有効な場合、本関数内で受信したデータを MIDI OUT 端子に接続されている機器に送信します。

3.10　R_MIDI_GetType()

MIDI 機器からの受信したメッセージの種別を受け取ります。

Format

```
fsp_err_t R_MIDI_GetType (
    midi_ctrl_t * const    p_api_ctrl,
    midi_type_t *          p_type
)
```

Parameters

p_api_ctrl

コントロール構造体へのポインタ

*p_type

受信メッセージの種別コードを格納する変数のポインタ

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

MIDI IN 端子に接続されている機器から受信した MIDI メッセージの種別データを取得し、引数 p_type に指定された領域に格納します。

Example

R_MIDI_Read()の Example を参照してください。

Special Notes

なし

3.11　R_MIDI_GetData1()

MIDI 機器から受信したメッセージの 1 番目のデータを取得します。

Format

```
fsp_err_t R_MIDI_GetData1 (
    midi_ctrl_t * const    p_api_ctrl,
    uint8_t *              p_data1
)
```

Parameters

p_api_ctrl

コントロール構造体へのポインタ

*p_data1

受信メッセージのデータを格納する変数のポインタ

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

MIDI IN 端子に接続されている機器から受信した MIDI メッセージの 1 番目のデータを取得し、引数 p_data1 に指定された領域に格納します。

Example

R_MIDI_Read()の Example を参照してください。

Special Notes

なし

3.12　R_MIDI_GetData2()

MIDI 機器から受信したメッセージの 2 番目のデータを取得します。

Format

```
fsp_err_t R_MIDI_GetData2 (
    midi_ctrl_t * const    p_api_ctrl,
    uint8_t *              p_data2
)
```

Parameters

p_api_ctrl

コントロール構造体へのポインタ

*p_data2

受信メッセージのデータを格納する変数のポインタ

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

MIDI IN 端子に接続されている機器から受信した MIDI メッセージの 2 番目のデータを取得し、引数 p_data2 に指定された領域に格納します。

Example

R_MIDI_Read()の Example を参照してください。

Special Notes

なし

3.13　R_MIDI_GetChannel()

MIDI 機器から受信したメッセージの MIDI チャンネル情報を受け取ります。

Format

```
fsp_err_t R_MIDI_GetChannel (
    midi_ctrl_t * const    p_api_ctrl,
    uint8_t *              p_channel
)
```

Parameters

p_api_ctrl

コントロール構造体へのポインタ

*p_channel

受信メッセージのデータを格納する変数のポインタ

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

MIDI IN 端子に接続されている機器から受信した MIDI メッセージのチャンネル情報を取得し、引数 p_channel に指定された領域に格納します。

受信した MIDI メッセージがチャンネルメッセージである場合、チャンネル情報(1~16)を格納します。

受信した MIDI メッセージがチャンネルメッセージでない場合、0 を格納します。

Example

R_MIDI_Read()の Example を参照してください。

Special Notes

なし

3.14　R_MIDI_SetInputChannel()

受信対象とする MIDI チャンネルを設定します。

Format

```
fsp_err_t R_MIDI_SetInputChannel (
    midi_ctrl_t * const   p_api_ctrl,
    uint8_t               channel
)
```

Parameters

p_api_ctrl

コントロール構造体へのポインタ

channel

受信対象とする MIDI チャンネル

MIDI_CHANNEL_OMNI : すべての MIDI チャンネルを入力対象にします
1 ~ 16 : 指定した MIDI チャンネルを入力対象にします
MIDI_CHANNEL_OFF : すべての MIDI チャンネルを入力禁止にします

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

受信対象とする MIDI チャンネルを設定します。

本関数は R_MIDI_Begin()関数を実行後に必要に応じて呼び出してください。

Example

```
midi_ctrl_t  midi_ctrl;

if (FSP_SUCCESS != R_MIDI_Begin(g_midi_c0_instance.p_ctrl, MIDI_CHANNEL_OMNI))
{
      /* Error */
}
else
{
   if (FSP_SUCCESS != R_MIDI_SetInputChannel(g_midi_c0_instance.p_ctrl, 1))
   {
         /* Error */
   }
```

Special Notes

なし

3.15　R_MIDI_SetThruFilterMode()

スルー機能を設定します。

Format

```
fsp_err_t R_MIDI_SetThruFilterMode (
    midi_ctrl_t * const    p_api_ctrl,
    midi_thru_mode_t    mode
)
```

Parameters

p_api_ctrl

コントロール構造体へのポインタ

mode

スルーモード

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

スルーモードを設定します。

本関数は R_MIDI_Open()関数を実行後に呼び出してください。

Example

```
midi_ctrl_t  midi_ctrl;

if (FSP_SUCCESS != R_MIDI_SetThruFilterMode (g_midi_c0_instance.p_ctrl,
MIDI_THRU_Off))
{
      /* Error */
}
```

Special Notes

スルー機能の初期値は MIDI_THRU_Full です。

3.16　R_MIDI_NotifyEvent()

UART 通信の送信および受信完了コールバック処理です。UART 通信の送信完了コールバック関数、または受信完了コールバック関数から本関数をコールしてください。

Format

```
fsp_err_t R_MIDI_NotifyEvent (
    midi_ctrl_t * const          p_api_ctrl,
    midi_peripheral_event_t      event
)
```

Parameters

p_api_ctrl

コントロール構造体へのポインタ

event

通知情報

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	引数のポインタが指定されていません
FSP_ERR_NOT_OPEN	Open()のコールなしにコールされました

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

UART の送信完了、または受信したことを本モジュールに通知します。

Example

「4.1 UART 通信機能の追加」を参照して下さい。

Special Notes

本関数は、UART 通信の送信完了コールバック関数、または受信完了コールバック関数からコールしてください。

3.17　R_MIDI_Notify1msCycle()

本モジュールのアクティブセンシング機能を有効にする場合、時間管理が必要になるためハードウェアタイマやソフトウェアタイマ等を使って、本関数を 1ms 間隔で定期的にコールする必要があります。

Format

```
fsp_err_t R_MIDI_Notify1msCycle (
        void
)
```

Parameters

なし

Return Values

FSP_SUCCESS	正常終了

Properties

r_midi_rl78_api.h にプロトタイプ宣言されています。

Description

本モジュールに 1ms 経過を通知します。

Example

「4.2 タイマ機能の追加」を参照して下さい。

Special Notes

時間管理が必要になるためハードウェアタイマやソフトウェアタイマ等を使って、本関数を 1ms 間隔で定期的にコールする必要があります。

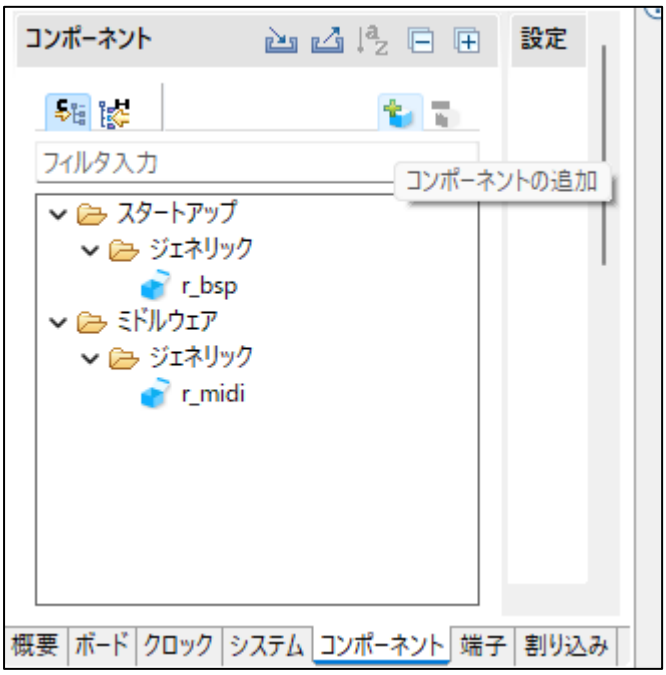
4.　本モジュール追加後の各種設定

本モジュールをプロジェクトに追加後、使用する環境に合わせて各コンポーネントの追加と設定、コンフィグレーションオプションを設定する必要があります。

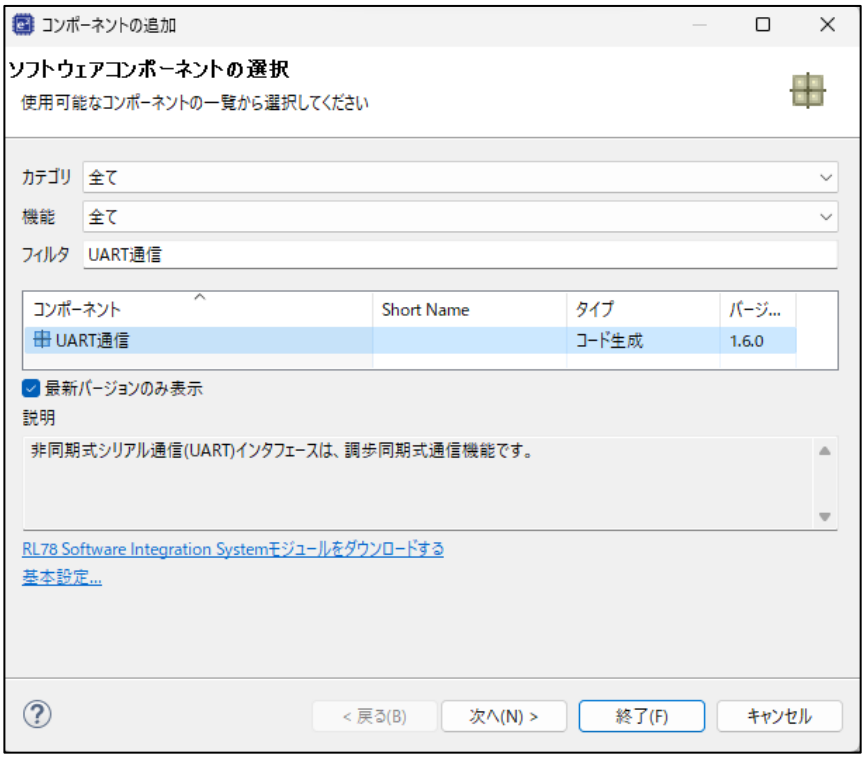
4.1　UART 通信機能の追加

4.1.1　UART コンポーネントの追加

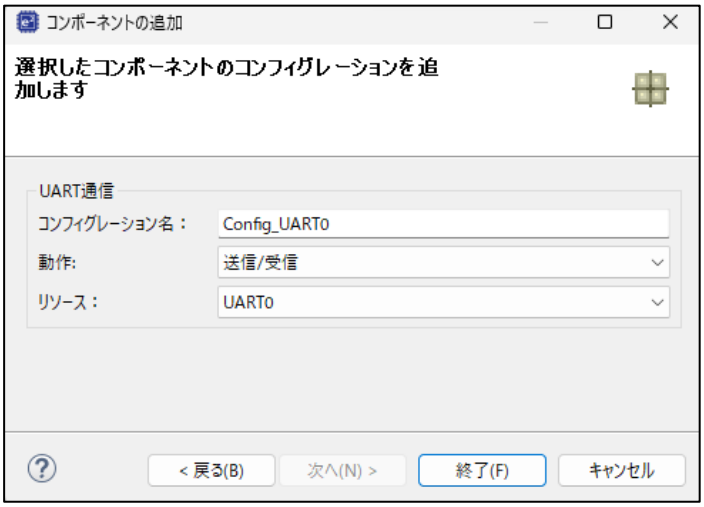
スマート・コンフィグレータ画面のコンポーネントタブから”コンポーネントの追加”アイコンをクリックします。



「UART 通信」を選択して「次へ」をクリックします。



コンフィグレーション名　　：任意の名前を設定してください。
※この名前を MIDI インタフェースモジュールのコンフィグレーション
“MIDI_CFG_C0_UART_COMPONENT”と一致させる必要があります。
動作　　：“送信/受信”を選択します。
リソース　　：外部機器と接続する UART のチャネルを選択してください。



「終了」をクリックします。

4.1.2　通信設定

(1)　シリアル・アレイ・ユニットの UART 通信を使用する場合

送信側は以下の通り設定します。

転送モード	: シングル転送モード
データ・ビット長	: 8 ビット
データ転送方向設定	: LSB
パリティ設定	: パリティ・ビットなし
ストップビット	: 1 ビット
送信データ・レベル設定	: 非反転(通常)
転送レート設定	: 31250 bps ※誤差が 5%以内になるようにクロック設定で調整してください。
割り込み設定	: 任意
コールバック機能設定	: ”送信完了”にチェックを入れてください。

続けて受信側は以下の通り設定します。
以下の通り設定します。

データ・ビット長	: 8 ビット
データ転送方向設定	: LSB
パリティ設定	: パリティ・ビットなし
受信データ・レベル設定	: 非反転(通常)
転送レート設定	: 31250 bps ※誤差が 5%以内になるようにクロック設定で調整してください。
割り込み設定(受信完了)	: 任意の割り込みレベルを設定して下さい。
割り込み設定(エラー)	: 本モジュールでは使用しません。必要に応じて有効にしてください。
コールバック機能設定	: ”受信完了”にチェックを入れてください。”受信エラー”は使用しません。

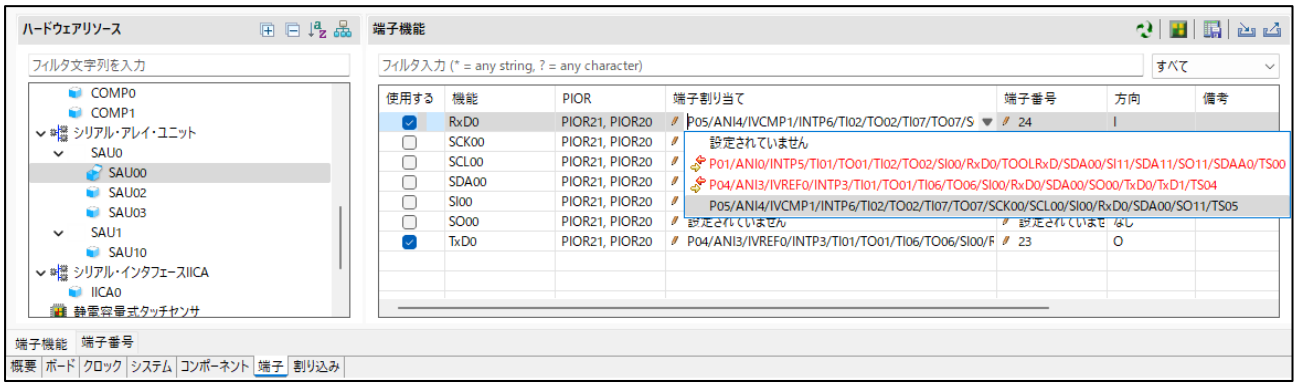
(2)　UARTA を使用する場合

以下の通り設定します。

CLKAn 端子出力設定	: 無効
データ・ビット長	: 8 ビット
データ転送方向設定	: LSB
パリティ設定	: パリティ・ビットなし
受信データ・レベル設定	: 非反転(通常)
送信モード設定	: 割り込みによる連続転送
受信エラーの設定	: 任意です。システムに合わせて設定してください。
転送レート設定	: 31250 bps ※誤差が 5%以内になるようにクロック設定で調整してください。
割り込み設定	: 任意の割り込みレベルを設定して下さい。
コールバック機能設定	: ”送信完了”と”受信完了”にチェックを入れてください。 ”受信エラー”は任意です。

4.1.3　端子設定

MCU によっては、１つの端子機能に対して複数の端子から選択する必要があります。
「端子」タブを表示した後、使用する UART チャネルの端子とポート番号を紐づけてください。



最後に、”コードの生成”ボタンを押してソースコードを生成します。

4.1.4　ソースコードの追加

生成した UART 通信コードの送信完了および受信完了の callback 内で本モジュールのイベント通知関数を呼び出すコードを追加します。

編集するファイル名は $(コンフィグレーション名)_user.c です。

① ヘッダファイル名の追加
#include "r_midi_rl78_if.h"
を追加します。
② 送信完了コールバック関数(関数名は r_$(コンフィグレーション名)_callback_sendend()) 内に
R_MIDI_NotifyEvent(g_midi_c0_instance.p_ctrl, MIDI_EVENT_UART_SEND);
を追加します。
③ 受信完了コールバック関数(関数名は r_$(コンフィグレーション名)_callback_receiveend())内に
R_MIDI_NotifyEvent(g_midi_c0_instance.p_ctrl, MIDI_EVENT_UART_RECV);
を追加します。

※コードの追加は、“Start user code for …”記載行と”End user code. …”記載行の間に記述して下さい。

Example

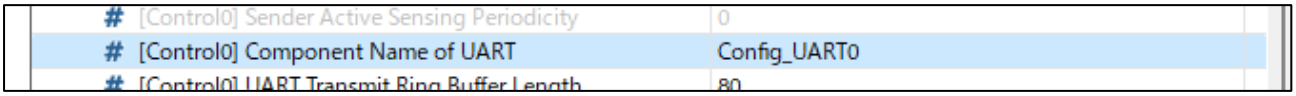
```
/* Start user code for include. Do not edit comment generated here */
#include "r_midi_rl78_if.h"
/* End user code. Do not edit comment generated here */

static void r_Config_UART0_callback_sendend(void)
{
    /* Start user code for r_Config_UART0_callback_sendend. Do not edit comment generated here */
    R_MIDI_NotifyEvent(g_midi_c0_instance.p_ctrl, MIDI_EVENT_UART_SEND);
    /* End user code. Do not edit comment generated here */
}

static void r_Config_UART0_callback_receiveend(void)
{
    /* Start user code for r_Config_UART0_callback_receiveend. Do not edit comment generated here */
    R_MIDI_NotifyEvent(g_midi_c0_instance.p_ctrl, MIDI_EVENT_UART_RECV);
    /* End user code. Do not edit comment generated here */
}
```

4.1.5　MIDI インタフェースモジュールとの紐づけ

MIDI インタフェースモジュールの“[Control0] Compornet Name of UART”の設定を、(1)で指定したコンフィグレーション名に合わせてください。



4.1.6　初期設定処理

UART の初期設定処理はスマート・コンフィグレータの機能により、MCU リセットから main 関数呼び出しされるまでに実行されます。そのため、ユーザ側で記述する必要はありません。

4.2　タイマ機能の追加

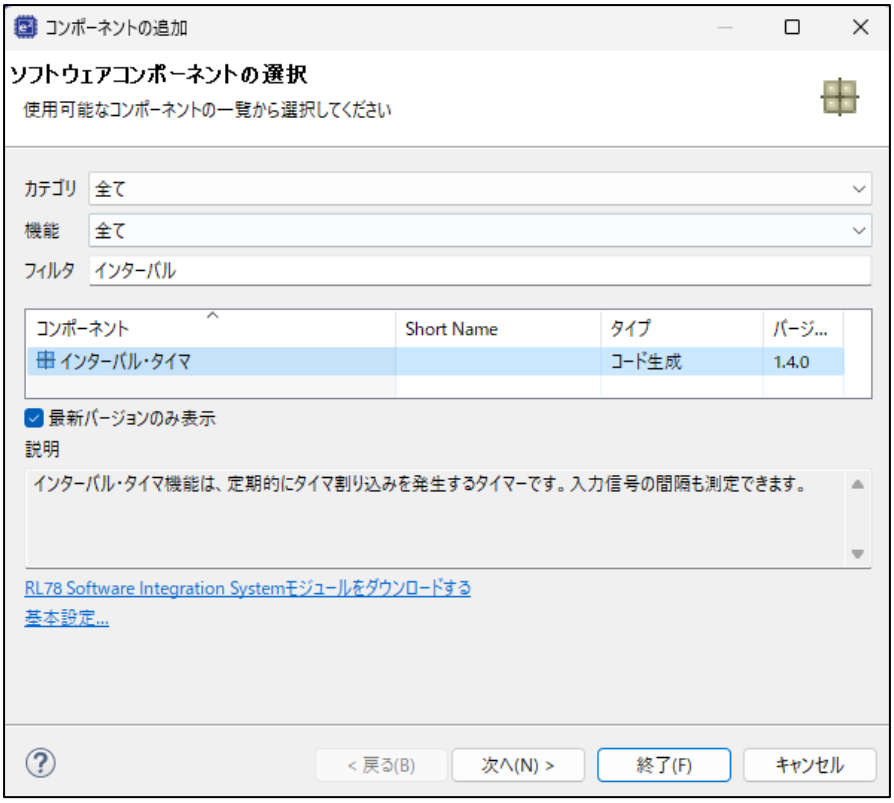
アクティブセンシング機能を有効にする場合、1ms 毎に API をコールして通知する必要があります。

ここでは MCU のタイマ機能を使った通知方法を説明します。

4.2.1　タイマコンポーネントの追加

スマート・コンフィグレータ画面の”コンポーネントの追加”画面から

「インターバル・タイマ」を選択して「次へ」をクリックします。



リソースは指定しないため、システムに応じて選択してください。

本資料では TAU0_0(16 ビット・カウンタ・モード)を用いて説明します。

「終了」をクリックします。

4.2.2　タイマ設定

下記を満たすように設定してください。

インターバル時間　　：1ms

割り込み設定　　：割り込み発生にチェックを入れてください。　割り込み優先度は任意です。

「コード生成」をクリックします。

4.2.3　タイマ割り込みでのソースコードの追加

生成したタイマ割り込みの callback 関数内に本モジュールのイベント通知関数を呼び出すコードを追加します。

① ヘッダファイルの追加

#include "r_midi_rl78_if.h"

を追加します。

② タイマ割り込みのコールバック関数内に

R_MIDI_Notify1msCycle();

を追加します。

Example

```
/* Start user code for include. Do not edit comment generated here */
#include "r_midi_rl78_if.h"
/* End user code. Do not edit comment generated here */


static void __near r_Config_TAU0_0_interrupt(void)
{
    /* Start user code for r_Config_TAU0_0_interrupt. Do not edit comment generated here */
    R_MIDI_Notify1msCycle();
    /* End user code. Do not edit comment generated here */
}
```

4.2.4　main プログラムでのソースコードの追加

生成したタイマの開始関数を main プログラムの冒頭に記述してください。

Example

```
#include "r_smc_entry.h"
#include "r_midi_rl78_if.h"

int main(void)
{
   R_Config_TAU0_0_Start();

   if (FSP_SUCCESS != R_MIDI_Open(g_midi_c0_instance.p_ctrl, g_midi_c0_instance.p_cfg))
   {
       /* Error */
   }
```

4.2.5　初期設定処理

タイマの初期設定処理はスマート・コンフィグレータの機能により、MCU リセットから main 関数呼び出しされるまでに実行されます。そのため、ユーザ側で記述する必要はありません。

5. 参考ドキュメント

ユーザーズマニュアル：ハードウェア
（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境
RL78 ファミリ CC-RL コンパイラ ユーザーズマニュアル（R20UT3123）
（最新版をルネサス エレクトロニクスホームページから入手してください。）

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2024.11.29	-	初版発行

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。