RL78 Family

MIDI Interface Module Software Integration System

Introduction

This application note describes the MIDI interface module (referred to as “this module hereafter”) that uses the Software Integration System (SIS). This module controls communication with MIDI devices by using the UART function of the serial array unit (SAU) in RL78 family MCUs from Renesas Electronics or by using the serial interface UARTA.

The following shows the position of this module.

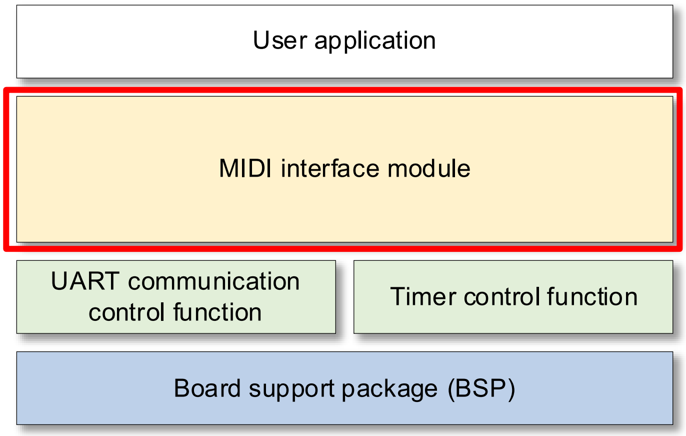

User application
MIDI interface module
UART communication control function
Timer control function
Board support package (BSP)

Figure 1 Position of This Module

This module complies with the MIDI 1.0 standard.

For details, visit the website of the Association of Musical Electronics Industry.

https://amei.or.jp/

Remark MIDI is a registered trademark of the Association of Musical Electronics Industry (AMEI)

Target Devices

・RL78 family

When applying this application note to other Renesas microcontrollers, modify it according to the specifications of the microcontroller and evaluate it thoroughly.

Related Documents

The following [1] and [2] are sample software that allows you to try out this module.

- [1] RL78 Family MIDI Linked Illumination Control Sample Software Using SIS (R01AN7463)
- [2] RL78 Family MIDI Performance Control Sample Software Using SIS (R01AN7491)
- [3] RL78 Family Board Support Package Module Using Software Integration System (R01AN5522)
- [4] Smart Configurator User's Manual: RL78 API Reference (R20UT4852)
- [5] RL78 Smart Configurator User's Guide: e^2 studio (R20AN0579)
- [6] RL78 Smart Configurator User's Guide: IAR (R20AN0581)
- [7] RL78/G16 Fast Prototyping Board User's Manual (R12UM0048)
- [8] Association of Musical Electronics Industry (rink)
- [9] Arduino MIDI Library (rink)

Contents

1. Overview

1.1 MIDI Interface Module

This module is MIDI control software that can be installed as an SIS module based on the Arduino MIDI Library.

It implements MIDI message input from the MIDI IN pin and MIDI message output to the MIDI OUT pin (the two pins are collectively referred to as “MIDI interface”).

Combined use of this module and Renesas software modules such as Smart Configurator code generation (available free of charge) enables control of MIDI-compatible devices. (See Figure 1-1 and Table 1-1.)

This module is installed in a project as an API. For details about how to install this module, see section 2.11 Adding the SIS Module.

Figure 1-1 shows the software configuration when this module is used for MIDI communication.

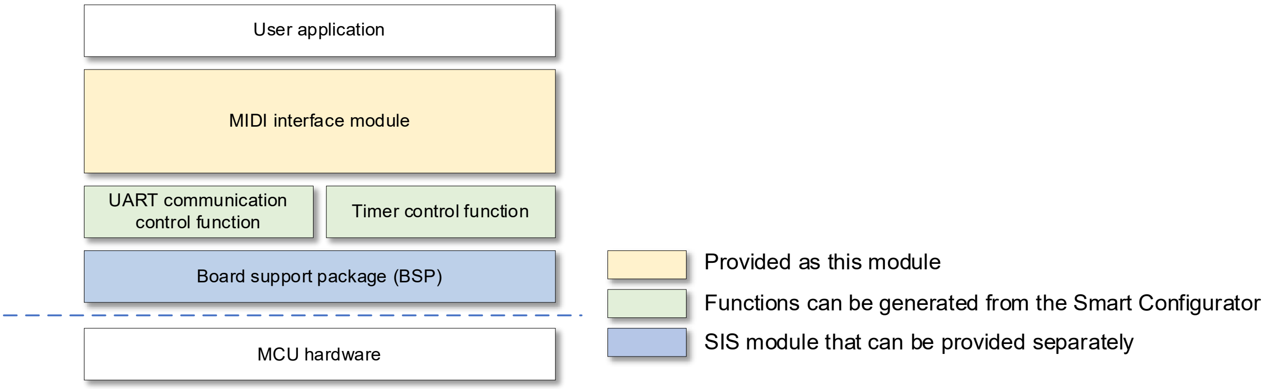

User application
MIDI interface module
UART communication control function
Timer control function
Board support package (BSP)
MCU hardware
Provided as this module
Functions can be generated from the Smart Configurator
SIS module that can be provided separately

Figure 1-1 Software Configuration

Table 1-1 lists the modules used in the application configuration.

Table 1-1 List of the Modules Used in the Application Configuration

Name	Component Name	Component Type
BSP (board support package module)	Board Support Package	RL78 Software Integration System
UART communication control	UART	Code generation
Timer control	Interval timer	Code generation

(1) Board support package module (BSP)

This component specifies initial settings such as clock setting. The settings are specified in the Smart Configurator.

(2) UART communication control

This component is used for MIDI communications with externally connected MIDI devices by using UART. The settings are specified in the Smart Configurator.

(3) Timer control

This component is used to manage elapsed time within this module. Use of this module is required depending on the configuration settings of this module. The settings are specified in the Smart Configurator.

1.2 Overview of the MIDI Interface Module

This module uses UART to control communication with MIDI devices.

Table 1-2 shows the functions of this module.

Table 1-2 List of MIDI Interface Module Functions

Item	Function
Base software	Arduino MIDI Library v5.0.2
Compliant standard	MIDI standard 1.0
Number of controllable MIDI interfaces	One (Specified in the configuration)
Active sensing function	Supported (Specified in the configuration)
Through function	Through output by software is supported. (The setting can be changed by API function call.)
Message receipt notification	Callback function registration and function calling according to messages are supported.
Reception filtering in units of MIDI channels	One of the following can be specified: ・Enable all channels for reception ・Enable only the specified channel for reception ・Disable all channels for reception (The setting can be changed by API function call.)
Standard MIDI file specification	Not supported

1.3 Hardware Settings

Figure 1-2 shows a hardware configuration example.

Connection with external devices uses commercially available MIDI cables.

MCU pin names vary depending on the serial interface to be used. See the pins and functions of the board to be used, and read the pin names as those of your MCU pins.

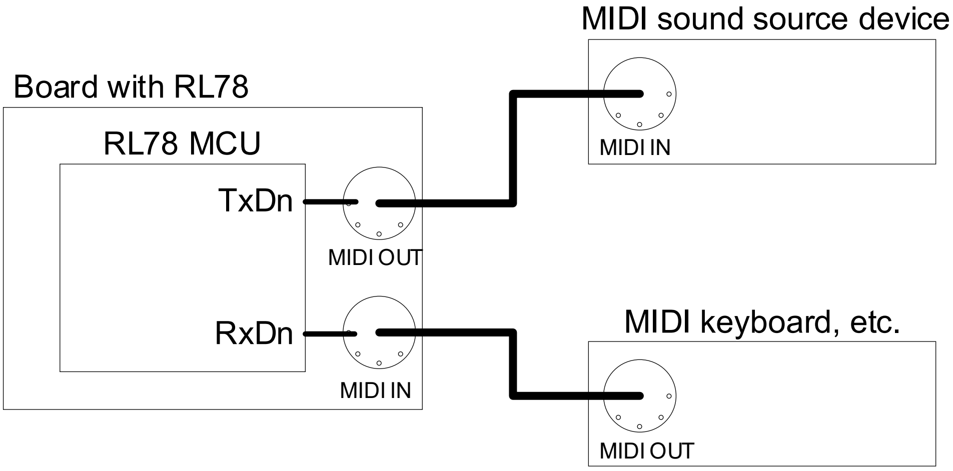


Figure 1-2 Hardware Configuration Example

Table 1-3 Used Pins and Functions

Pin Name	I/O	Description
TxDn	Output	UART data transmission
RxDn	Input	UART data reception

1.4 API Overview

Table 1-4 lists the API functions included in this module.

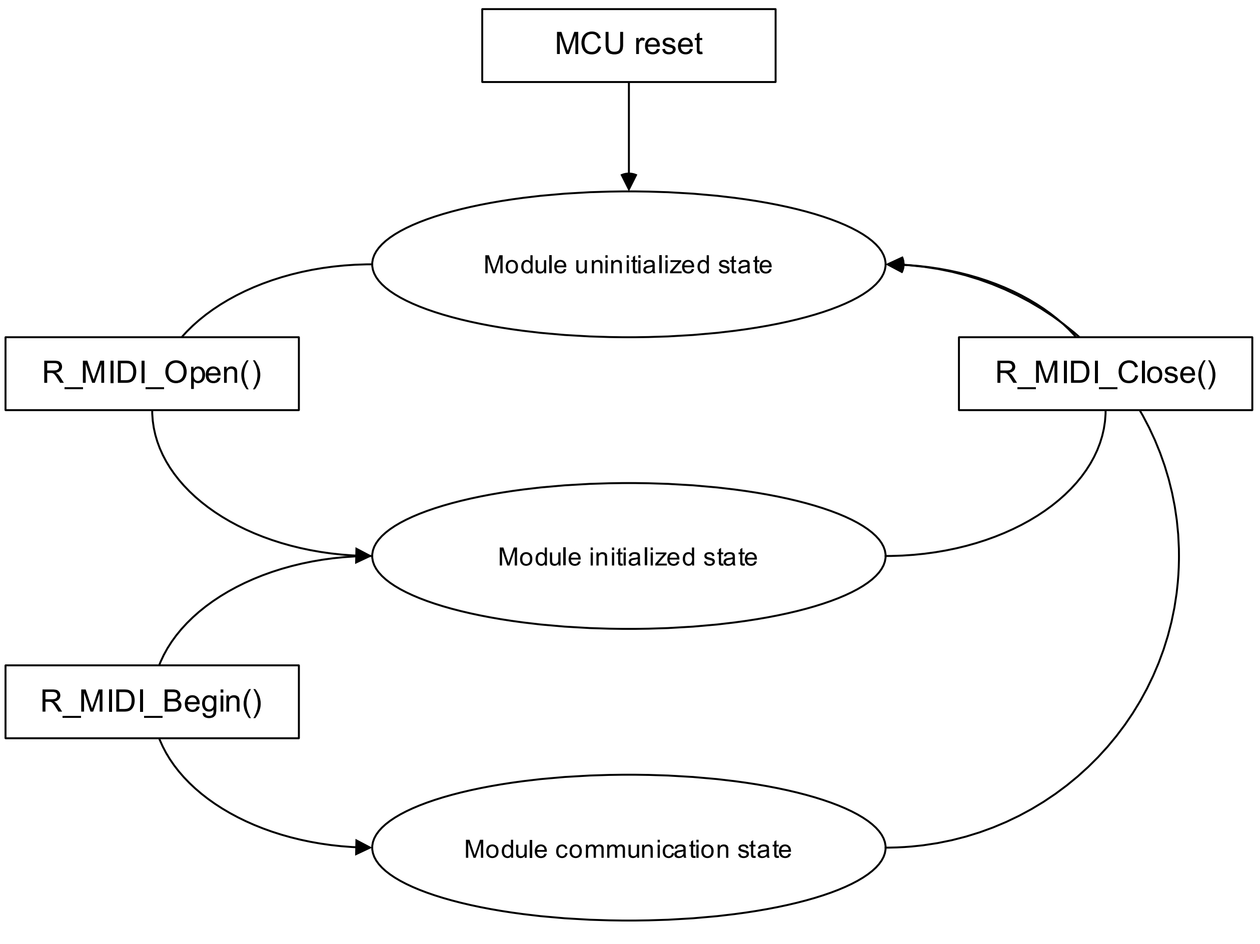
Table 1-4 List of API Functions

Function	Description
R_MIDI_Open()	Module open processing
R_MIDI_Close()	Module close processing
R_MIDI_Begin()	Module communication start processing
R_MIDI_SendNoteOn()	NoteOn message transmission processing
R_MIDI_SendNoteOff()	NoteOff message transmission processing
R_MIDI_SendProgramChange()	ProgramChange message transmission processing
R_MIDI_SendControlChange()	ControlChange message transmission processing
R_MIDI_Send()	Message transmission common processing
R_MIDI_Read()	Message reception processing
R_MIDI_GetType()	Processing to acquire type information for a received message
R_MIDI_GetChannel()	Processing to acquire channel information for a received message
R_MIDI_GetData1()	Processing to acquire data 1 of a received message
R_MIDI_GetData2()	Processing to acquire data 2 of a received message
R_MIDI_SetInputChannel()	Reception-target channel set processing
R_MIDI_SetThruFilterMode()	Through mode set processing
R_MIDI_NotifyEvent()	Callback processing for notification of transmission or reception in UART communication
R_MIDI_Notify1msInterval() (Note 1)	Interval timer counter processing

Note 1: When the active sensing function is enabled, this function must be called at 1-ms intervals by using a hardware timer or software timer for time management.

1.5 State Transition Diagram

Figure 1-3 shows the state transition diagram for this module.



Note: The state does not change if an API function not shown in the figure is executed.

Figure 1-3 State Transition Diagram for the MIDI Interface Module

2. API Information

Operation of this module has been confirmed under the following conditions.

2.1 Hardware Requirements

Your MCU must support one the following functions:

- CSI(UART)
- UARTA

2.2 Software Requirements

This module depends on the following module.

- Board support package (r_bsp) Rev.1.62 or later

It also depends on the following code generation module.

- UART

2.3 Supported Tool Chains

Operation of this module has been confirmed with the following tool chains:

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

2.4 Interrupt Vector to Use

None

2.5 Header Files

All API calls and supporting interface definitions are defined as follows.

r_midi_rl78_if.h	: The user must include this file when using this module.
r_midi_rl78_api.h	: This file contains the API definitions.
r_midi_rl78_config.h	: This file is used to set configuration options.

2.6 Integer Types

This driver uses ANSI C99. These types are defined in stdint.h.

2.7 Settings at Compilation

The configuration options for this module are set in r_midi_rl78_config.h.

When using the Smart Configurator, the configuration options can be set in the software component configuration screen. The set values are automatically applied to r_midi_rl78_config.h when the module is added. The following table describes the option names and settings.

Configuration options in r_midi_rl78_config.h	
MIDI_CFG_PARAM_CHECKING_ENABLE Note: The default value is "BSP_CFG_PARAM_CHECKING_ENABLE".	1: Parameter check processing is included in the code during build. 0: Parameter check processing is omitted from the code during build. If BSP_CFG_PARAM_CHECKING_ENABLE is specified for this option, the system default setting is used.
MIDI_CFG_CTRL_NUM_MAX Note: The default value is "1".	Define the number of MIDI interfaces. Note: Only 1 can be specified in this version.
MIDI_CFG_C0_USE_RUNNING_STATUS Note: The default value is "false".	Define whether to omit transmission of status bytes when the same status message is sent consecutively. true: Omit false: Do not omit
MIDI_CFG_C0_HNDL_NULL_VELOCITY Note: The default value is "true".	Define the notification method to the user when a NoteOn message with velocity = 0 is received. true: The message is handled as a NoteOff message. false: The message is handled as a NoteOn message.
MIDI_CFG_C0_USE_1BYTE_PARSING Note: The default value is "true".	Specify the receive data analysis method when R_MIDI_Read() is executed. true: Each time R_MIDI_Read() is executed, only 1 byte of receive data is analyzed before processing ends. false: All data accumulated in the receive buffer is analyzed each time R_MIDI_Read() is executed.
MIDI_CFG_C0_SYSEX_MAX_SIZE Note: The default value is "128".	Define the size of the buffer that stores SystemExclusive messages. Specify an even value in the range from 4 to 10.
MIDI_CFG_C0_USE_SND_ACTSENSE Note: The default value is "false".	Specify whether to send an active sensing message to the MIDI OUT pin on a regular basis. true: Send false: Do not send
MIDI_CFG_C0_ USE_REV_ACTSENSE Note: The default value is "false".	Specify whether to monitor active sensing messages received from the device connected to the MIDI IN pin. true: Monitor false: Do not monitor
MIDI_CFG_C0_SND_ACTSENSE_PERIOD Note: The default value is "0".	Specify the transmission interval of active sensing messages, in the range from 0 to 250. If 0 is specified, an active sensing message is not sent.
MIDI_CFG_C0_UART_COMPONENT Note: The default value is "Config_UART0".	Define the configuration name of UART communication that performs MIDI communication. This definition is used for generating function names for UART communication specified for UART communication of the Smart Configurator.
MIDI_CFG_C0_UART_TXBUF_SIZE Note: The default value is "80".	Define the buffer size for transmission in MIDI communication. Specify an even value in the range from 4 to 1024.

Configuration options in r_midi_rl78_config.h	
MIDI_CFG_C0_UART_RXBUF_SIZE Note: The default value is ”80”.	Define the buffer size for reception in MIDI communication. Specify an even value in the range from 4 to 1024.

2.8 Code Size

The following table shows the ROM size, RAM size, and maximum available stack size for this module.

The sizes of ROM (code and constants) and RAM (global data) are determined by the configuration options during build described in section 2.7 Settings at Compilation.

The values in the table below have been confirmed under the following conditions.

Module revision: r_midi_rl78 rev.1.00

Compiler versions: Renesas Electronics RL78 Family C Compiler Package V1.13.00
(Default settings of the integrated development environment)
IAR C/C++ Compiler for Renesas RL78 version 5.10.3
(Default settings of the integrated development environment)

Configuration options: Default settings

ROM, RAM, and Stack Code Sizes

Device	Category	Memory Used					
		Renesas Compiler				IAR Compiler	
		Optimization level: -Olite		Optimization level: -Odefault		Optmization level (-Og) Low level	
		Parameter check		Parameter check		Parameter check	
		Provided	None	Provided	None	proveided	None
RL78/ G16	ROM	6619	6086	5986	5497	6755	5848
	RAM	461		461		419	
	Maximum available stack size	90		88		86	

2.9 Arguments

This section describes the structures used as arguments of API functions. These structures are described in r_midi_rl78_api.h along with the prototype declarations of the API functions.

(1) Control structure definition

This is a control structure of the MIDI interface configuration. There is no need to specify the settings from the application.

A variable appropriate for the configuration of this module is output by generating code from the Smart Configurator. Specify the p_ctrl member of this variable for the first argument of the API of this module.

```
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) Configuration structure definition

A variable appropriate for the configuration of this module is output by generating code from the Smart Configurator. Specify the p_cfg member of this variable for the second argument of the API of this module.

```
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;
```

The following shows the structure used in the preceding structure.

(3) midi_settings_t structure definition

```
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) Macros

The following shows the macros and enumerated types used by this module.

・midi_type_t type

```
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_type

```
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 type

```
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 Return Values

This section shows the return values of API functions. This enumerated type is described in fsp_common_api.h of the board support package.

```
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 Adding the SIS Module

This module must be added for each project to be used.

(1) Adding the SIS module by using the Smart Configurator in e^2 studio
Use the Smart Configurator in e^2 studio to automatically add the SIS module to the user project. For details, refer to the related document [3].

(2) Adding the SIS module by using the Smart Configurator in IAREW
Use the Smart Configurator Standalone version to automatically add the SIS module to the user project. For details, refer to the related document [4].

3. API Functions

3.1 R_MIDI_Open()

This function initializes this module. This function must be executed before using any other API function.

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

Pointer to the control structure

p_cfg

Pointer to the configuration structure

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_ALREADY_OPEN	Open() is called without calling Close().
FSP_ERR_INVALID_ARGUMENT	A configuration parameter is invalid.

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function initializes the control structure area specified by the argument p_api_ctrl, and then links that area with the working area specified by the argument p_cfg.

The control structure and working area must be retained and their contents must not be modified by using an application program before the close processing ends.

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

The initial setting of the UART function is required before this function is executed. For details, see section 4.1 Adding UART Communication .

3.2 R_MIDI_Close()

This function closes this module.

Format

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

Parameters

p_api_ctrl

Pointer to the control structure

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function terminates processing of this module, and then release the resource.

After execution of this function, the control structure specified by the argument or various working areas specified in the configuration in R_MIDI_Open() are no longer used. They can be used for other purposes.

Example

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

Special Notes

None

3.3 R_MIDI_Begin()

This function starts communication with external MIDI devices.

Format

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

Parameters

p_api_ctrl

Pointer to the control structure

channel

Reception target MIDI channels

MIDI_CHANNEL_OMNI : All MIDI channels are selected for input target.

1 to 16 : The specified MIDI channel is selected for input target.

MIDI_CHANNEL_OFF : All MIDI channels are disabled for input.

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function starts MIDI communication with the connected device.

When data is received on the MIDI channel specified by the argument, the result can be acquired by using the R_MIDI_Read() function.

The reception target MIDI channels can be changed by executing R_MIDI_SetInputChannel() after this function is executed.

Example

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

Special Notes

None

3.4 R_MIDI_SendNoteOn()

This function sends a NoteOn message.

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

Pointer to the control structure

inNoteNumber

Note number (0 to 127)

inVelocity :

Velocity (0 to 127)

inChannel :

MIDI channel (1 to 16)

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function sends a NoteOn message to the device connected to the MIDI OUT pin.

Execute this function after executing R_MIDI_Begin().

Example

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

Special Notes

The values of lower 7 bits are valid in the inNoteNumber and inVelocity arguments.

3.5 R_MIDI_SendNoteOff()

This function sends a NoteOff message.

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

Pointer to the control structure

inNoteNumber

Note number (0 to 127)

inVelocity :

Velocity (0 to 127)

inChannel :

MIDI channel (1 to 16)

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function sends a NoteOff message to the device connected to the MIDI OUT pin.

Execute this function after executing R_MIDI_Begin().

Example

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

Special Notes

The values of lower 7 bits are valid in the inNoteNumber and inVelocity arguments.

3.6 R_MIDI_SendProgramChange()

This function sends a ProgramChange message.

Format

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

Parameters

p_api_ctrl

Pointer to the control structure

inProgramNumber

Program number (0 to 127)

inChannel :

MIDI channel (1 to 16)

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function sends a ProgramChange message to the device connected to the MIDI OUT pin.

Execute this function after executing R_MIDI_Begin().

Example

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

Special Notes

The values of lower 7 bits are valid in the inProgramNumber argument.

3.7 R_MIDI_SendControlChange()

This function sends a ControlChange message.

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

Pointer to the control structure

inControlNumber

Control number (0 to 127)

inControlValue

Set value (0 to 127)

inChannel :

MIDI channel (1 to 16)

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function sends a ControlChange message to the device connected to the MIDI OUT pin.

Execute this function after executing R_MIDI_Begin().

Example

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

Special Notes

The values of lower 7 bits are valid in the inProgramNumber argument.

3.8 R_MIDI_Send()

This function sends a message to a MIDI device.

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

Pointer to the control structure

inType

Transmission message type

The range of messages that can be specified is from MIDI_TYPE_NoteOff to MIDI_TYPE_PitchBend, or from MIDI_TYPE_Clock to MIDI_TYPE_SystemReset.

inData1

Set value 1

inData2

Set value 2

inChannel :

MIDI channel (1 to 16)

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function sends a message to the device connected to the MIDI OUT pin.

Execute this function after executing R_MIDI_Begin().

Example

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

Special Notes

To specify a message selected from MIDI_TYPE_Clock to MIDI_TYPE_SystemReset for inType, specify 0 for the inData1, inData2, and inChannel arguments.

3.9 R_MIDI_Read()

This function receives a message receipt notification from the MIDI device.

Format

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

Parameters

p_api_ctrl

Pointer to the control structure

*p_result

Pointer to the variable that stores the result of message receipt notification.

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function checks whether a message is received from the device connected to the MIDI IN pin, and then stores the result in the area specified by p_result.

For the result value, “true” indicates that a message is received and “false” indicates that no message is received.

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

When through mode is enabled, the data received in this function is sent to the device connected to the MIDI OUT pin.

3.10 R_MIDI_GetType()

This function receives the type of the message received from the MIDI device.

Format

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

Parameters

p_api_ctrl

Pointer to the control structure

*p_type

Pointer to the variable that stores the type code of the received message

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function acquires the type data of the MIDI message received from the device connected to the MIDI IN pin, and then stores that data in the area specified by the argument p_type.

Example

See “Example” in “R_MIDI_Read()”.

Special Notes

None

3.11 R_MIDI_GetData1()

This function acquires the first data of the message received from the MIDI device.

Format

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

Parameters

p_api_ctrl

Pointer to the control structure

*p_data1

Pointer to the variable that stores the received message data

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function acquires the first data of the MIDI message received from the device connected to the MIDI IN pin, and then stores that data in the area specified by the argument p_data1.

Example

See “Example” in “R_MIDI_Read()”.

Special Notes

None

3.12 R_MIDI_GetData2()

This function acquires the second data of the message received from the MIDI device.

Format

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

Parameters

p_api_ctrl

Pointer to the control structure

*p_data2

Pointer to the variable that stores the received message data

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function acquires the second data of the MIDI message received from the device connected to the MIDI IN pin, and then stores that data in the area specified by the argument p_data2.

Example

See “Example” in “R_MIDI_Read()”.

Special Notes

None

3.13 R_MIDI_GetChannel()

This function acquires MIDI channel information for a message received from the MIDI device.

Format

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

Parameters

p_api_ctrl

Pointer to the control structure

*p_channel

Pointer to the variable that stores the received message data

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function acquires channel information for the MIDI message received from the device connected to the MIDI IN pin, and then stores it in the area specified by the argument p_channel.

If the received MIDI message is a channel message, channel information (1 to 16) is stored.

If the received MIDI message is not a channel message, 0 is stored.

Example

See “Example” in “R_MIDI_Read()”.

Special Notes

None

3.14 R_MIDI_SetInputChannel()

This function specifies MIDI channels for reception target.

Format

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

Parameters

p_api_ctrl

Pointer to the control structure

channel

Reception target MIDI channels

MIDI_CHANNEL_OMNI : All MIDI channels are selected for input target.

1 to 16 : The specified MIDI channel is selected for input target.

MIDI_CHANNEL_OFF : All MIDI channels are disabled for input.

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function specifies MIDI channels for reception target.

Call this function if necessary after executing the R_MIDI_Begin() function.

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

None

3.15 R_MIDI_SetThruFilterMode()

This function sets the through function.

Format

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

Parameters

p_api_ctrl

Pointer to the control structure

mode

Through mode

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function sets the through mode.

Call this function after executing the R_MIDI_Open() function.

Example

```
midi_ctrl_t  midi_ctrl;

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

Special Notes

The initial value of the through function is MIDI_THRU_Full.

3.16 R_MIDI_NotifyEvent()

This function performs callback processing for completion of transmission and reception in UART communication. Call this function from the transmission end callback function or reception end callback function for UART communication.

Format

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

Parameters

p_api_ctrl

Pointer to the control structure

event

Notification information

Return Values

FSP_SUCCESS	Normal end
FSP_ERR_ASSERTION	The pointer of the argument is not specified.
FSP_ERR_NOT_OPEN	The function is called without calling Open().

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function notifies this module of completion of UART transmission or reception.

Example

See section 4.1 Adding UART Communication .

Special Notes

Call this function from the transmission end callback function or reception end callback function in UART communication.

3.17 R_MIDI_Notify1msCycle()

When the active sensing function of this module is enabled, this function must be called at 1-ms intervals by using a hardware timer or software timer for time management.

Format

```
fsp_err_t R_MIDI_Notify1msCycle (
        void
)
```

Parameters

None

Return Values

FSP_SUCCESS	Normal end

Properties

The prototype declaration is contained in r_midi_rl78_api.h.

Description

This function notifies this module of the elapsed time (1 ms).

Example

See section 4.2 Adding the Timer Function.

Special Notes

This function must be called at 1-ms intervals by using a hardware timer or software timer for time management.

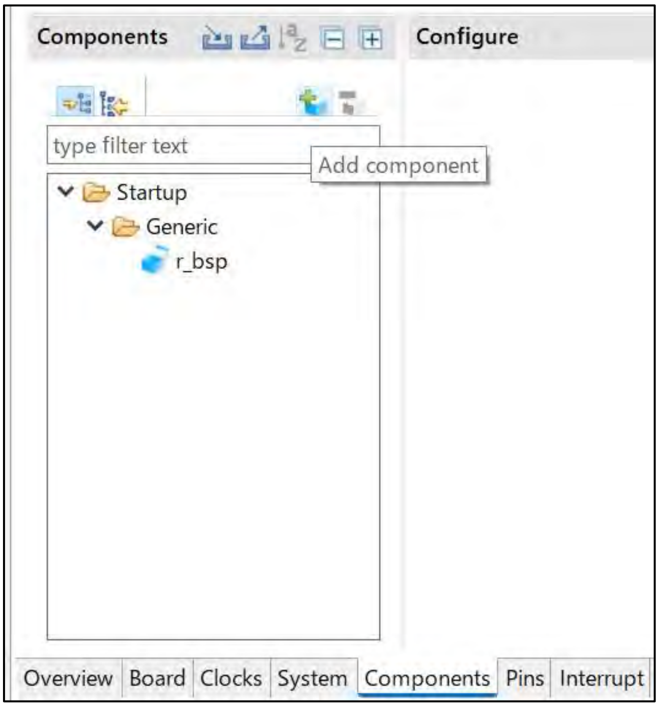
4. Various Settings after Adding This Module

After adding this module to your project, you must add and configure components and specify configuration options according to the environment to be used.

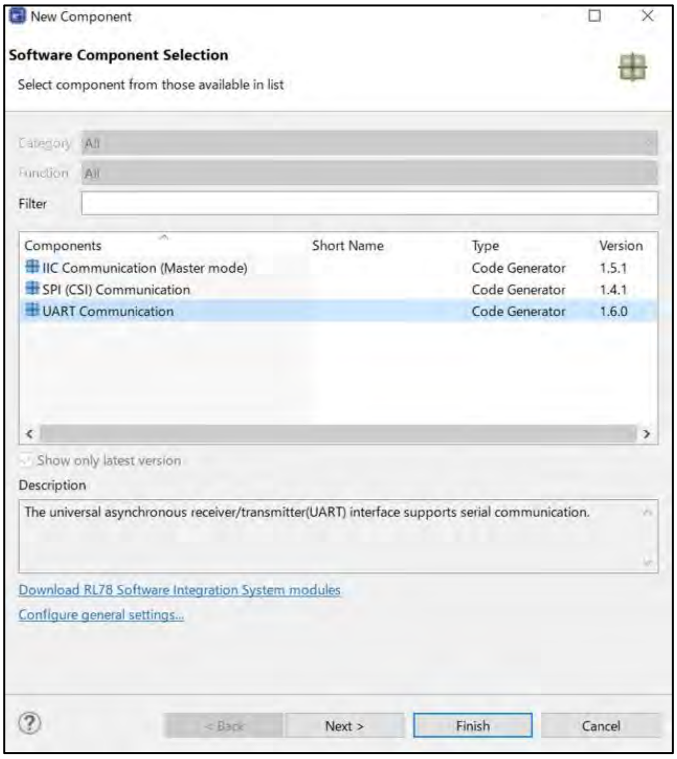
4.1 Adding UART Communication Functions

4.1.1 Adding the UART component

Click [Add component] on the [Components] tab of the Smart Configurator window.



Select [UART Communication], and then click [Next].



Software Component Selection

Select component from those available in list

Category All

Function All

Filter

Components	Short Name	Type	Version
IIC Communication (Master mode)		Code Generator	1.5.1
SPI (CSI) Communication		Code Generator	1.4.1
UART Communication		Code Generator	1.6.0

□ Show only latest version

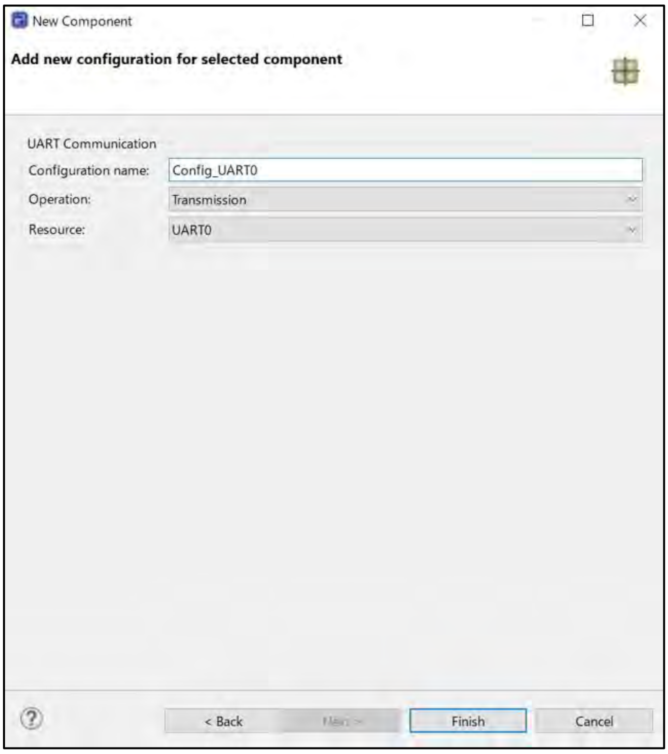
Description

The universal asynchronous receiver/transmitter(UART) interface supports serial communication.

Download RL78 Software Integration System modules

Configure general settings...

Configuration name	: Specify any name. Note: This name must match the setting in the configuration "MIDI_CFG_C0_UART_COMPONENT" of the MIDI interface module.
Operation	: Select [Transmission].
Resource	: Select the UART channel connected to the external device.



UART Communication

Configuration name: Config_UART0

Operation: Transmission

Resource: UART0

Click [Finish].

4.1.2 Communication settings

(1) When using UART communication for the serial array unit

Specify the transmission settings as follows:

Transfer mode setting : Single transfer mode

Data length setting : 8 bits

Transfer direction setting : LSB

Parity setting : None

Stop bit length setting : 1 bit

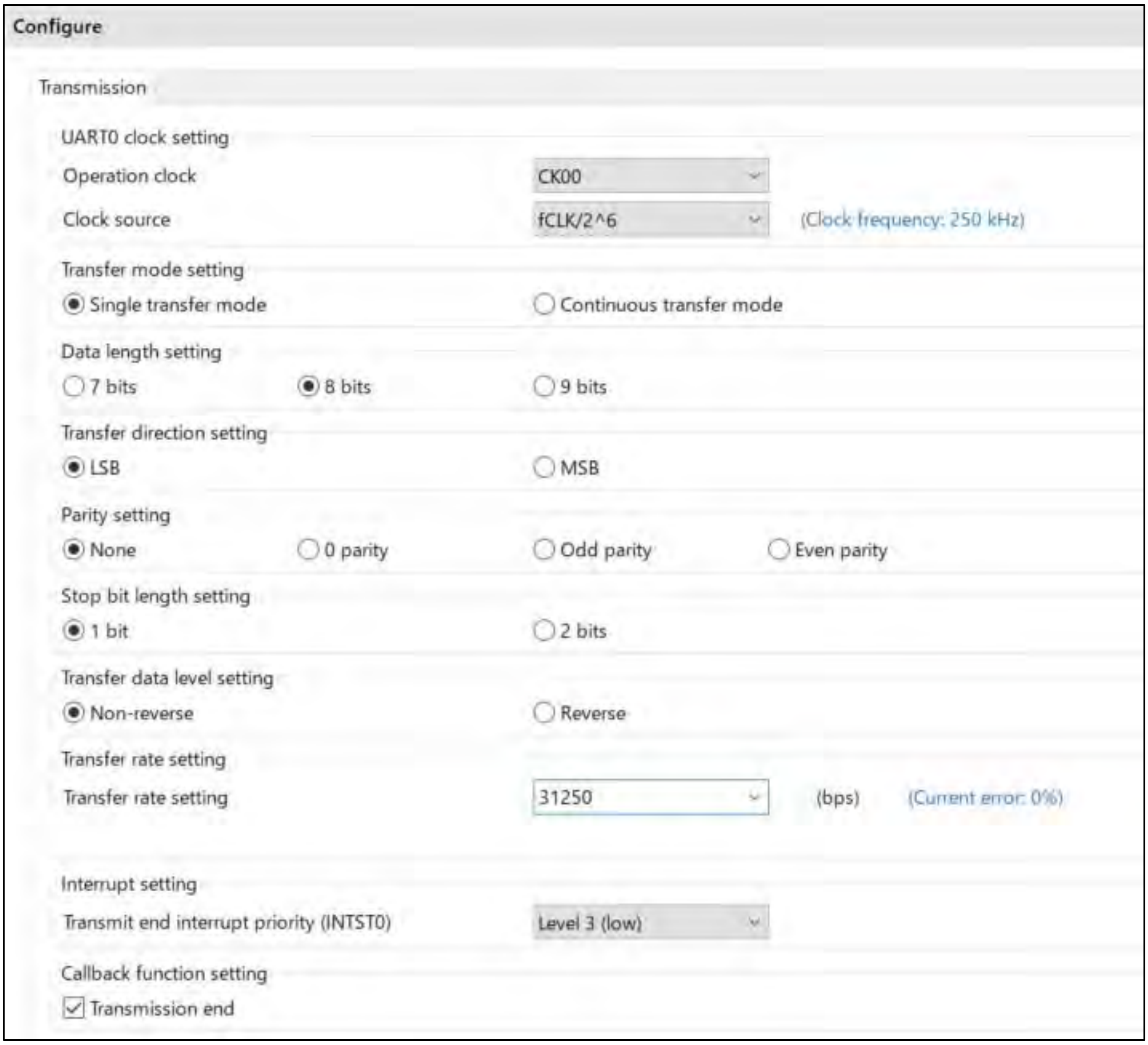
Transfer data level setting : Non-reverse

Transfer rate setting : 31250 (bps)

Note: Adjust the clock setting so that the error is within 5%.

Interrupt setting : Optional

Callback function setting : Select the [Transmission end] check box.



Configure

Transmission

UART0 clock setting

Operation clock CK00

Clock source fCLK/2^6 (Clock frequency: 250 kHz)

Transfer mode setting

● Single transfer mode ○ Continuous transfer mode

Data length setting

○ 7 bits ● 8 bits ○ 9 bits

Transfer direction setting

● LSB ○ MSB

Parity setting

● None ○ 0 parity ○ Odd parity ○ Even parity

Stop bit length setting

● 1 bit ○ 2 bits

Transfer data level setting

● Non-reverse ○ Reverse

Transfer rate setting

Transfer rate setting 31250 (bps) (Current error: 0%)

Interrupt setting

Transmit end interrupt priority (INTST0) Level 3 (low)

Callback function setting

☑ Transmission end

Then, specify the reception settings as follows:

Data length setting : 8 bits

Transfer direction setting : LSB

Parity setting : None

Receive data level setting : Non-reverse

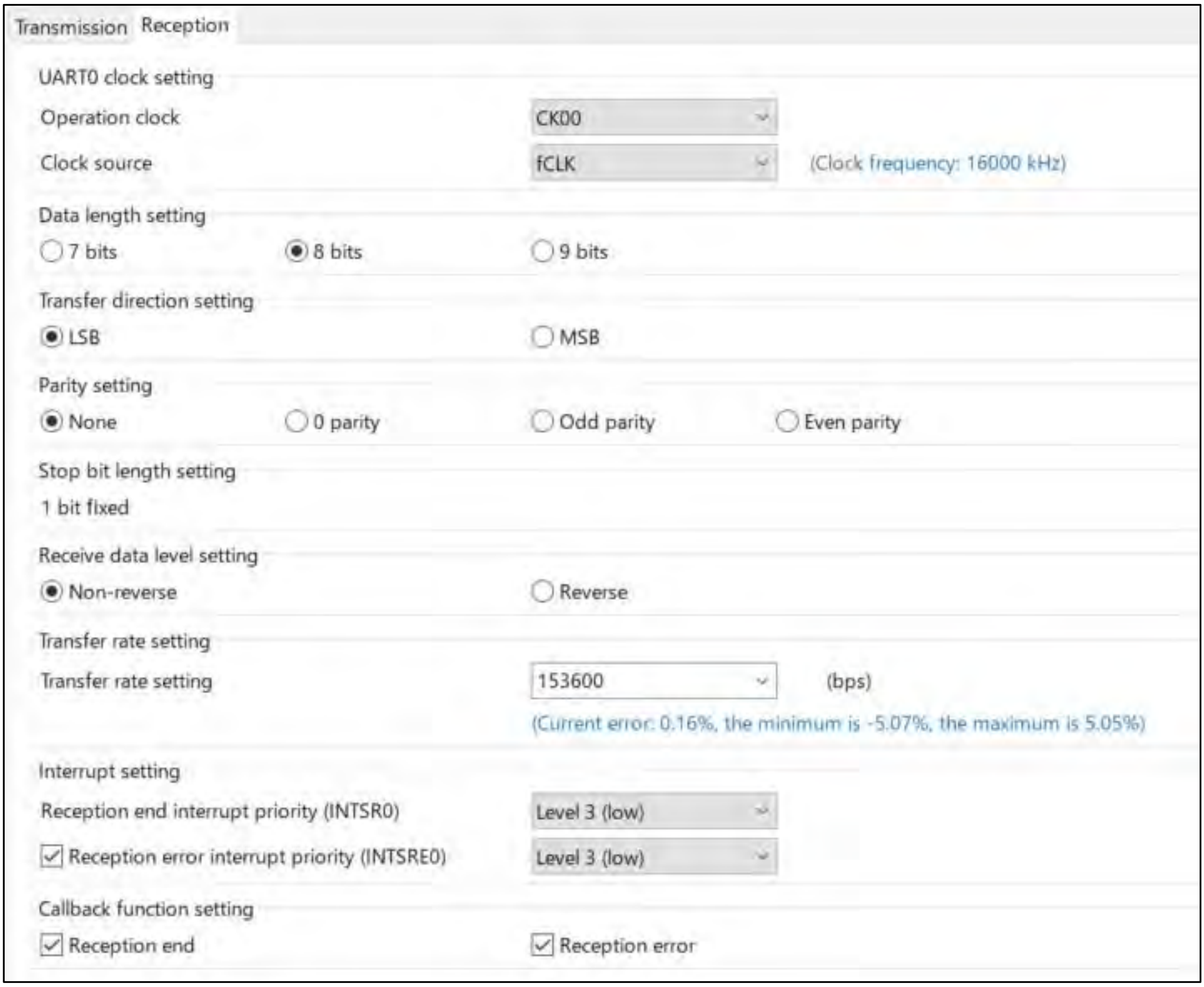
Transfer rate setting : 31250 bps

Note: Adjust the clock setting so that the error is within 5%.

Interrupt setting (Reception end): Specify any interrupt level.

Interrupt setting (Error) : This is not used in this module. Select the check box if necessary.

Callback function setting : Select the [Reception end] check box. The [Reception error] check box is not used.



Transmission Reception

UART0 clock setting

Operation clock CK00

Clock source fCLK (Clock frequency: 16000 kHz)

Data length setting

○ 7 bits ● 8 bits ○ 9 bits

Transfer direction setting

● LSB ○ MSB

Parity setting

● None ○ 0 parity ○ Odd parity ○ Even parity

Stop bit length setting

1 bit fixed

Receive data level setting

● Non-reverse ○ Reverse

Transfer rate setting

Transfer rate setting 153600 (bps)

(Current error: 0.16%, the minimum is -5.07%, the maximum is 5.05%)

Interrupt setting

Reception end interrupt priority (INTSR0) Level 3 (low)

☑ Reception error interrupt priority (INTSRE0) Level 3 (low)

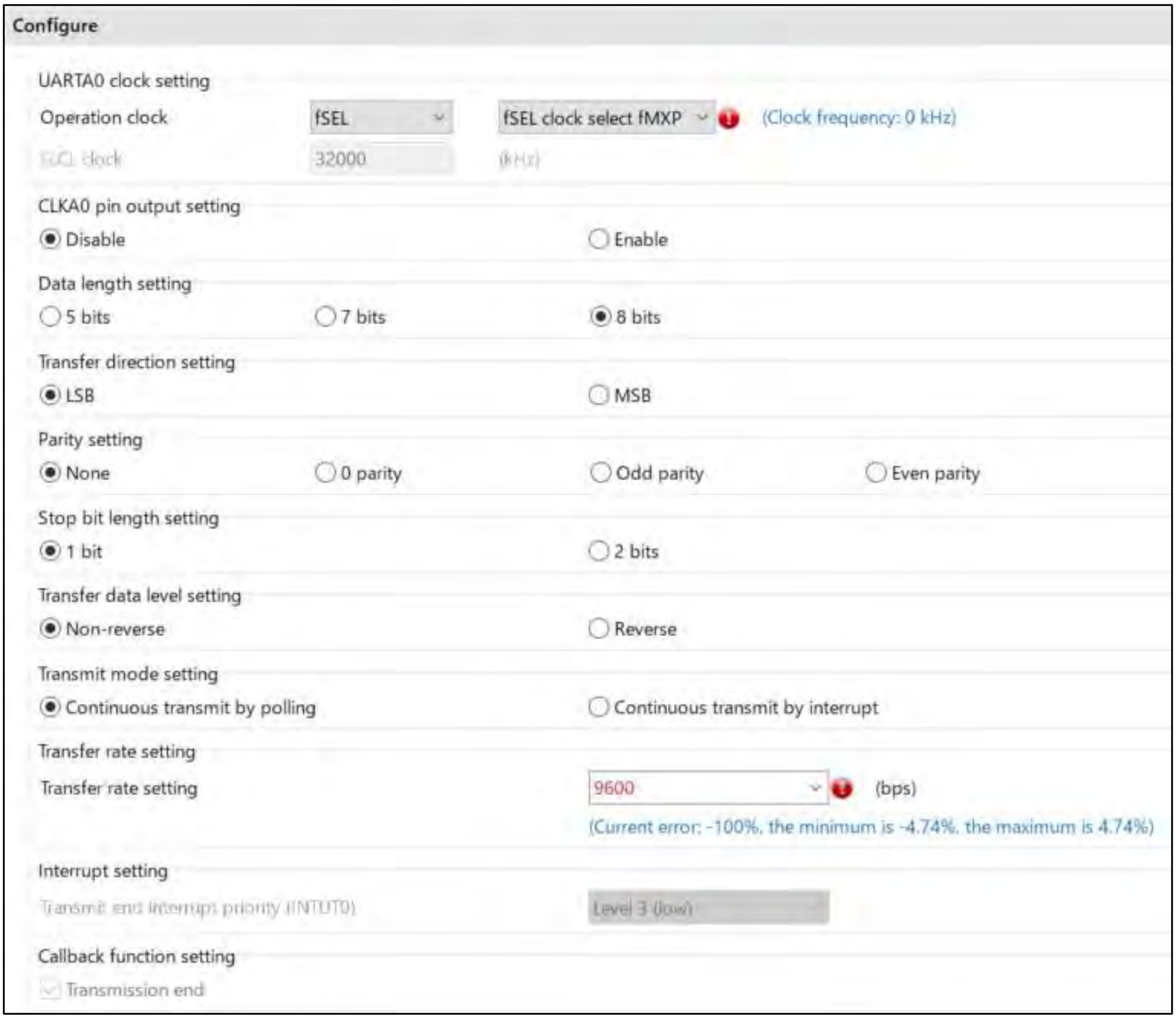
Callback function setting

☑ Reception end ☑ Reception error

(2) When using UARTA

Specify as follows:

CLKAn pin output setting	: Disable
Data length setting	: 8 bits
Transfer direction setting	: LSB
Parity setting	: None
Receive data level setting	: Non-reverse
Transmit mode setting	: Continuous transmit by interrupt
Receive error setting	: This setting is optional. Specify the appropriate setting for the system.
Transfer rate setting	: 31250 bps Note: Adjust the clock setting so that the error is within 5%.
Interrupt setting	: Specify any interrupt level.
Callback function setting	: Select the [Transfer end] and [Reception end] check boxes. Select the [Reception error] check box as desired.



Configure

UARTA0 clock setting

Operation clock: fSEL | fSEL clock select fMXP (Clock frequency: 0 kHz)

fSEL clock: 32000 (kHz)

CLKA0 pin output setting

● Disable ○ Enable

Data length setting

○ 5 bits ○ 7 bits ● 8 bits

Transfer direction setting

● LSB ○ MSB

Parity setting

● None ○ 0 parity ○ Odd parity ○ Even parity

Stop bit length setting

● 1 bit ○ 2 bits

Transfer data level setting

● Non-reverse ○ Reverse

Transmit mode setting

● Continuous transmit by polling ○ Continuous transmit by interrupt

Transfer rate setting

Transfer rate setting: 9600 (bps)

(Current error: -100%, the minimum is -4.74%, the maximum is 4.74%)

Interrupt setting

Transmit end interrupt priority (INTUT0): Level 3 (low)

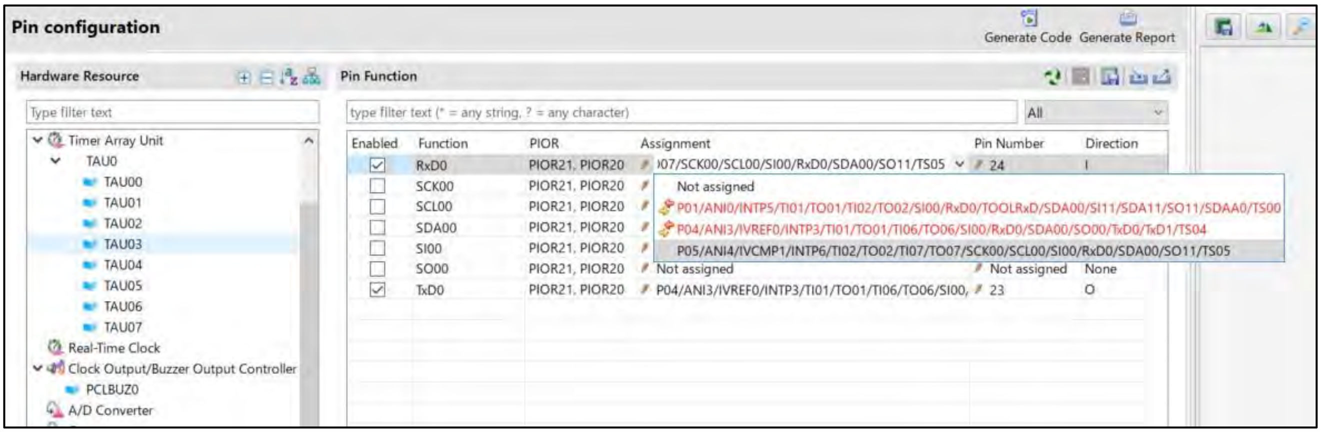
Callback function setting

☑ Transmission end

4.1.3 Pin settings

Depending on the MCU, one pin function may need to be selected from multiple pins.

On the [Pins] tab, link the enabled pins of the UART channel with the port numbers.



Enabled	Function	PIOR	Assignment	Pin Number	Direction
☑	RxD0	PIOR21, PIOR20	)07/SCK00/SCL00/SI00/RxD0/SDA00/SO11/TS05	24	I
☐	SCK00	PIOR21, PIOR20			
☐	SCL00	PIOR21, PIOR20			
☐	SDA00	PIOR21, PIOR20			
☐	SI00	PIOR21, PIOR20			
☐	SO00	PIOR21, PIOR20	Not assigned	Not assigned	None
☑	TxD0	PIOR21, PIOR20	P04/ANI3/IVREF0/INTP3/TI01/TO01/TI06/TO06/SI00,	23	O

Not assigned
P01/ANI0/INTP5/TI01/TO01/TI02/TO02/SI00/RxD0/TOOLRxD/SDA00/SI11/SDA11/SO11/SDAA0/TS00
P04/ANI3/IVREF0/INTP3/TI01/TO01/TI06/TO06/SI00/RxD0/SDA00/SO00/TxD0/TxD1/TS04
P05/ANI4/IVCMP1/INTP6/TI02/TO02/TI07/TO07/SCK00/SCL00/SI00/RxD0/SDA00/SO11/TS05

Finally, click [Generate Code] to generate the source code.

4.1.4 Adding a source code

Add the code that calls the event notification function of this module in the transmission end and reception end callback functions of the generated UART communication code.

The name of the file to be edited is $(*configuration-name*)_user.c.

① Adding the header file name
Add the following:
#include "r_midi_rl78_if.h"

② Add the following in the transmission end callback function (the function name is r_$(*configuration-name*)_callback_sendend()):
R_MIDI_NotifyEvent(g_midi_c0_instance.p_ctrl, MIDI_EVENT_UART_SEND);

③ Add the following in the reception end callback function (the function name is r_$(*configuration-name*)_callback_receiveend()):
R_MIDI_NotifyEvent(g_midi_c0_instance.p_ctrl, MIDI_EVENT_UART_RECV);

Note: The code must be added between the line starting with “Start user code for …” and the line starting with ”End user code. …”.

Example

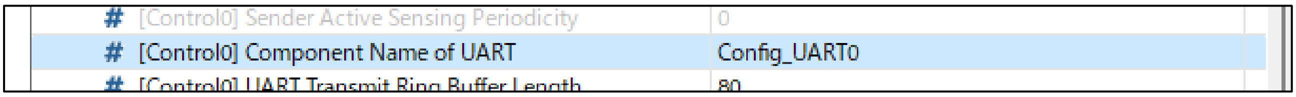
```
/* Start user code for include. Do not edit comment generated here */
#include "r_midi_rl78_if.h"
/* End user code. Do not edit comment generated here */

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 Linking with the MIDI interface module

Make sure that the setting of “[Control0] Component Name of UART” of the MIDI interface module matches the configuration name you specified in (1).



# [Control0] Sender Active Sensing Periodicity	0
# [Control0] Component Name of UART	Config_UART0
# [Control0] UART Transmit Ring Buffer Length	80

4.1.6 Initial setting processing

The user does not need to describe initial setting processing for UART, which is performed by the Smart Configurator function after the MCU is reset and before the main function is called.

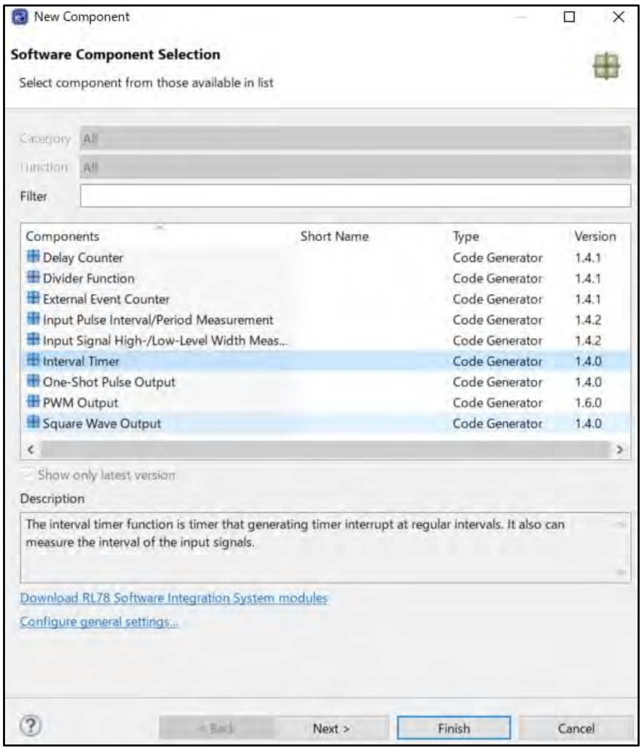
4.2 Adding the Timer Function

When the active sensing function is enabled, the API must be called at 1-ms intervals for notification.

This section describes how to send a notification by using the timer function of the MCU.

4.2.1 Adding the timer component

In the New Component window of the Smart Configurator, select [Interval Timer], and then click [Next].



Software Component Selection

Select component from those available in list

Category: All

Function: All

Filter:

Components	Short Name	Type	Version
Delay Counter		Code Generator	1.4.1
Divider Function		Code Generator	1.4.1
External Event Counter		Code Generator	1.4.1
Input Pulse Interval/Period Measurement		Code Generator	1.4.2
Input Signal High-/Low-Level Width Meas...		Code Generator	1.4.2
Interval Timer		Code Generator	1.4.0
One-Shot Pulse Output		Code Generator	1.4.0
PWM Output		Code Generator	1.6.0
Square Wave Output		Code Generator	1.4.0

Show only latest version

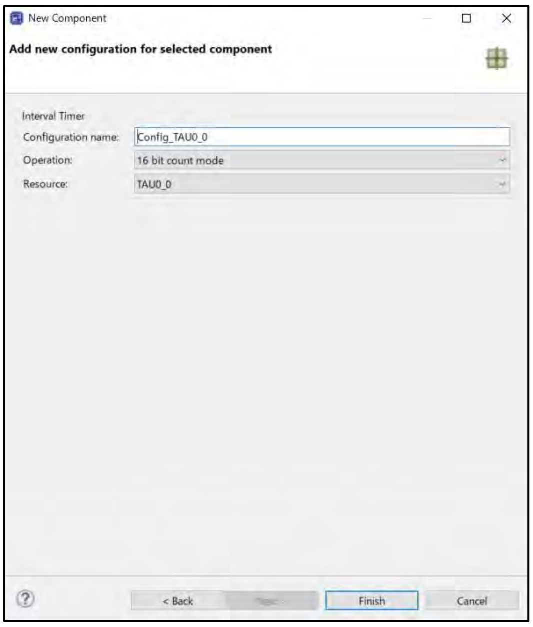
Description

The interval timer function is timer that generating timer interrupt at regular intervals. It also can measure the interval of the input signals.

Download RL78 Software Integration System modules

Configure general settings...

Because no resources are specified, select the appropriate option for the system.
This document describes the procedure by using TAU0_0 (16 bit counter mode).



Add new configuration for selected component

Interval Timer

Configuration name: Config_TAU0_0

Operation: 16 bit count mode

Resource: TAU0_0

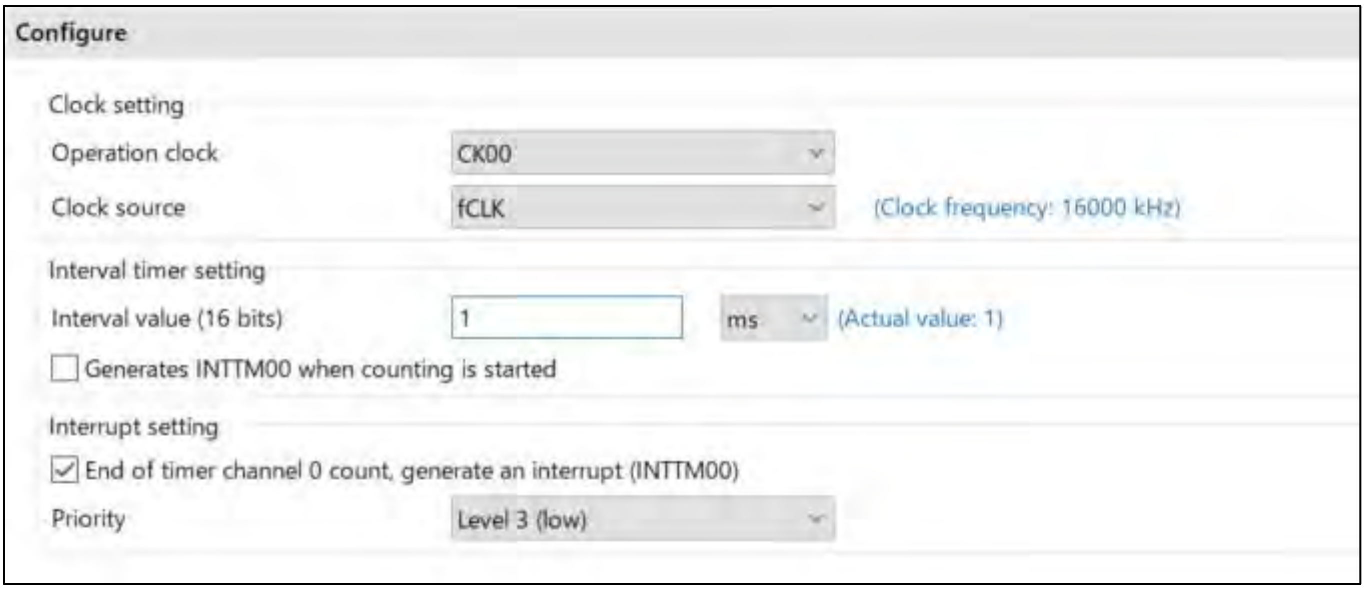
Click [Finish].

4.2.2 Timer settings

Specify the settings as follows.

Interval value : 1 ms

Interrupt setting: Select the [Generate an interrupt] check box. The interrupt priority is optional.



Configure

Clock setting

Operation clock: CK00

Clock source: fCLK (Clock frequency: 16000 kHz)

Interval timer setting

Interval value (16 bits): 1 ms (Actual value: 1)

□ Generates INTTM00 when counting is started

Interrupt setting

☑ End of timer channel 0 count, generate an interrupt (INTTM00)

Priority: Level 3 (low)

Click [Generate Code].

4.2.3 Adding a source code for timer interrupt

Add the code that calls the event notification function of this module in the timer interrupt callback function you generated.

① Adding the header file

Add the following:

#include "r_midi_rl78_if.h"

② Add the following to the timer interrupt callback function:

R_MIDI_Notify1msCycle();

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 Adding a source code in the main program

Add the generated timer start function in the beginning of the main program.

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 Initial setting processing

The user does not need to describe initialization processing for the timer, which is performed by the Smart Configurator function after the MCU is reset and before the main function is called.

5. Reference Documents

User’s Manual: Hardware
（The latest versions can be downloaded from the Renesas Electronics website.）

Technical Update / Technical News
（The latest versions can be downloaded from the Renesas Electronics website.）

User's Manual: Development Environment
RL78 Family CC-RL Compiler User's Manual（R20UT3123）
（The latest versions can be downloaded from the Renesas Electronics website.）

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	2024.11.29	-	First Edition

All trademarks and registered trademarks are the property of their respective owners.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.