Renesas Sample Demo Software

USB Speaker Demo Software for RX231HMI Kit

R01AN3828EJ0100
Rev.1.00
Apr 26, 2017

Summary

This application note describes the operation and functionality of USB Speaker Demo Software for RX231 HMI Kit (“the software”), which is audio playback software employing the RX231 that provides support for USB audio device class 1.0.

Target Board

R0K5RX231D000BR (RX231 HMI Solution Kit)

Web page

< https://www.renesas.com/products/software-tools/boards-and-kits/evaluation-demo-solution-boards/r0k5rx231d000br-rx231-hmi-solution-kit.html >

Contents

1. Introduction

The USB functionality of the RX231 supports isochronous data transfer. In isochronous transfer, data is transferred at fixed intervals, and no retry is attempted when a data transfer error occurs. This makes it suitable for applications where real-time performance is more important than data accuracy, such as audio or video playback.

USB Speaker Demo Software for RX231 HMI Kit, which is described in this application note, provides speaker functionality that utilizes isochronous out transfer as specified in the USB audio device class 1.0 standard. Therefore, when the RX231 HMI Kit, programmed with the software, is connected to a speaker with integrated amplifier or to headphones, it functions as a “USB speaker” outputting audio from the USB host.

Note: The content of this document comprises reference examples based on the USB standard, but their operation in an actual system is not guaranteed. When considering the incorporation of sample code into your system, make sure to carefully consider the system as a whole. The final decision on whether or not to incorporate sample code is the responsibility of the user.

Note: USB audio device classes are defined in the USB standard to enable control of the audio, voice, and music functions of embedded devices. They include capabilities for controlling transfer of audio data as well as functions, such as volume or tone, that directly affect the music environment.

(a) **Terms and Abbreviations**

ADCD:	Audio device class Driver
API:	Application program interface
APL:	Application
CS+:	Renesas integrated development environment
e^2 studio:	Eclipse embedded studio
FIT:	Firmware integration technology
LCD:	Liquid crystal display
LPC:	Low power consumption
RX231HMI Kit:	RX231 human machine interface solution kit
SSI:	Serial sound interface
USB:	Universal serial bus
USB Basic Mini FIT:	USB basic mini host and peripheral driver firmware integration technology

(b) **Related Documents**

Table 1.1 Related Documents

No.	Document Title	Revision	Issuer
1.	Universal Serial Bus Specification	2.0	USB-IF
2.	Universal Serial Bus Device Class Definition for Audio Devices	1.0	
3.	Universal Serial Bus Device Class Definition for Terminal Types	1.0	
4.	Universal Serial Bus Device Class Definition for Audio Data Formats	1.0	
5.	PCM1774 Data Sheet	—	Texas Instruments
6.	RX Family Board Support Package Module Using Firmware Integration Technology (Document No.: R01AN1685EJ)	3.30	Renesas Electronics
7.	USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) Using Firmware Integration Technology (Document No.: R01AN2166EJ)	1.02	
8.	SSI Module using Firmware Integration Technology (Document No.: R01AN2150EJ)	1.20	
9.	Simple I^{2}C Module Using Firmware Integration Technology (Document No.: R01AN1691EJ)	1.60	
10.	LPC Module Using Firmware Integration Technology (Document No.: R01AN2769EJ)	1.40	
11.	RX231 Group User's Manual: Hardware (Document No.: R01UH0496EJ)	1.10	
12.	RX231 Group Human Machine Interface Solution Kit R0K5RX231D000BR (Document No.: R01AN2586EJ)	1.02	
13.	RX231 HMI Solution Kit Base Demo Software (Function limited edition) RTK5RX2310P000F0ZR (Document No.: R11AN0015EJ)	1.00	

Note: USB-IF <http://www.usb.org/developers/docs/>

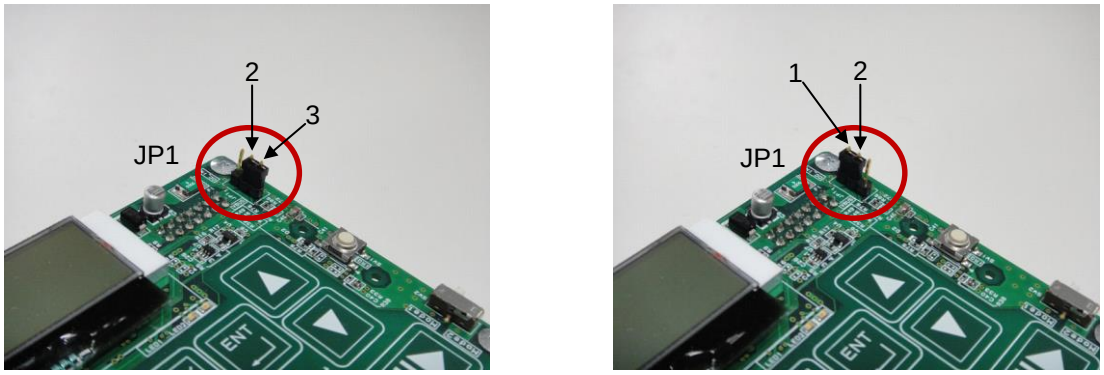
2. Usage

This section explains how to use the software. For detailed specifications of the RX231 HMI Kit, refer to item No. 12 in Table 1.1, Related Documents.

Follow the procedure below to prepare the board:

1. Write the software to the RX231 HMI Kit using an integrated development environment (e^2 studio or CS+) or Renesas Flash Programmer.

2. As shown in Figure 2.1, short JP1 pins 2 and 3 for bus-powered operation. For self-powered operation, short JP1 pins 1 and 2.



(a) Bus-Powered
JP1 Pins 2 and 3 Shorted

(b) Self-Powered
JP1 Pins 1 and 2 Shorted

Figure 2.1 RX231 HMI Kit JP1 Jumper Settings

Note: Renesas Flash Programmer
< https://www.renesas.com/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html>

3 As shown in Figure 2.2 and Figure 2.3, connect a speaker with integrated amplifier or headphones to the J2 jack, and connect the USB host to the USB micro-B connector. Do not connect J3 to a speaker or headphones.

4. Once connected, the USB host installs the necessary drivers automatically.

5. The setup is ready to use if the LCD backlight of the RX231 HMI Kit turns on and "RENESAS RX231 USB Audio Sample" is displayed.

6. Sound should issue from the speaker or headphones when music playback is initiated on the USB host. If no sound is audible, confirm that "speaker (USB Audio Sample)" is selected as the playback device in Microsoft Windows.

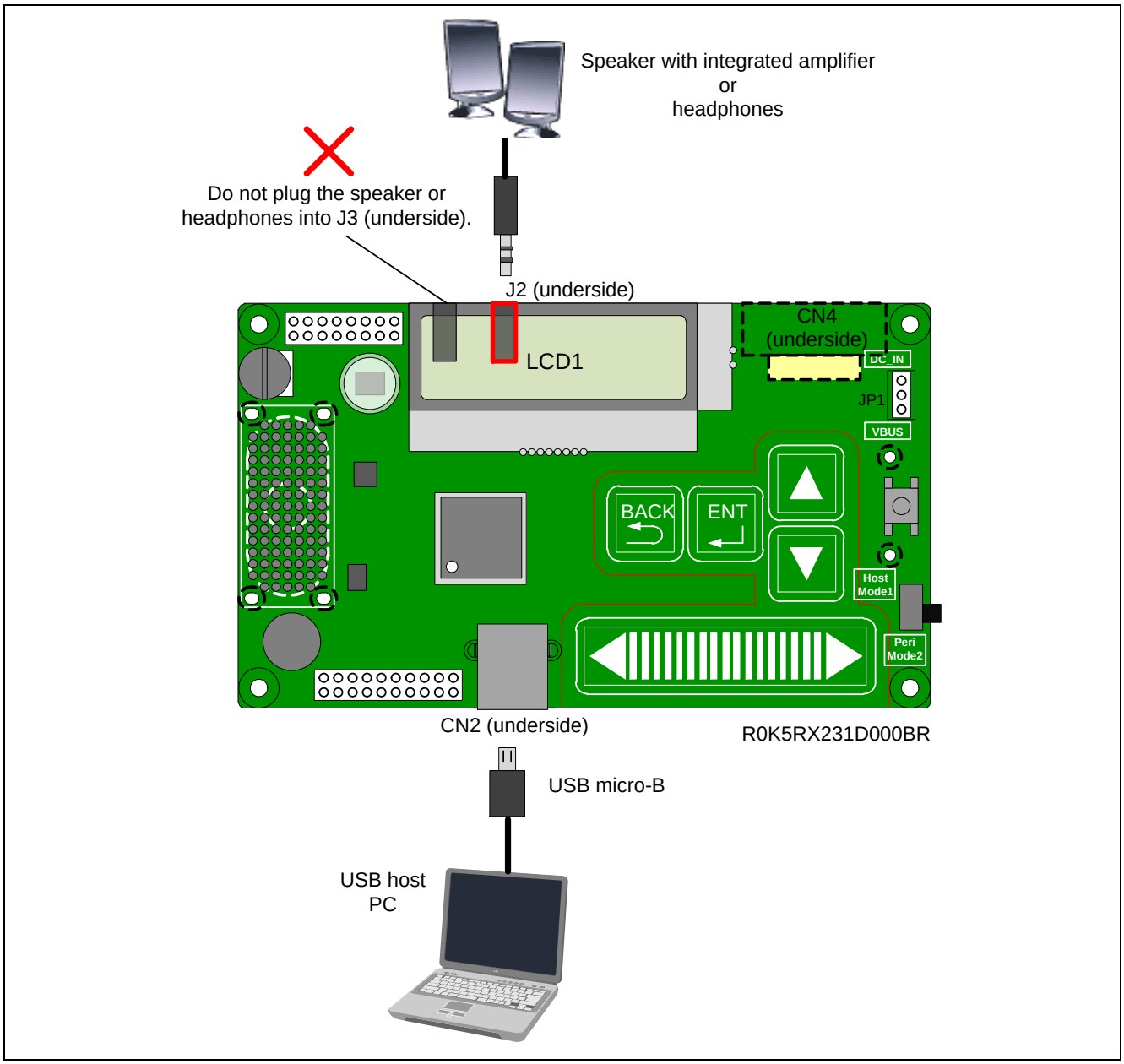


Figure 2.2 Connections of RX231 HMI Kit, USB host, and Speaker with Integrated Amplifier or Headphones (Bus-Powered Operation)

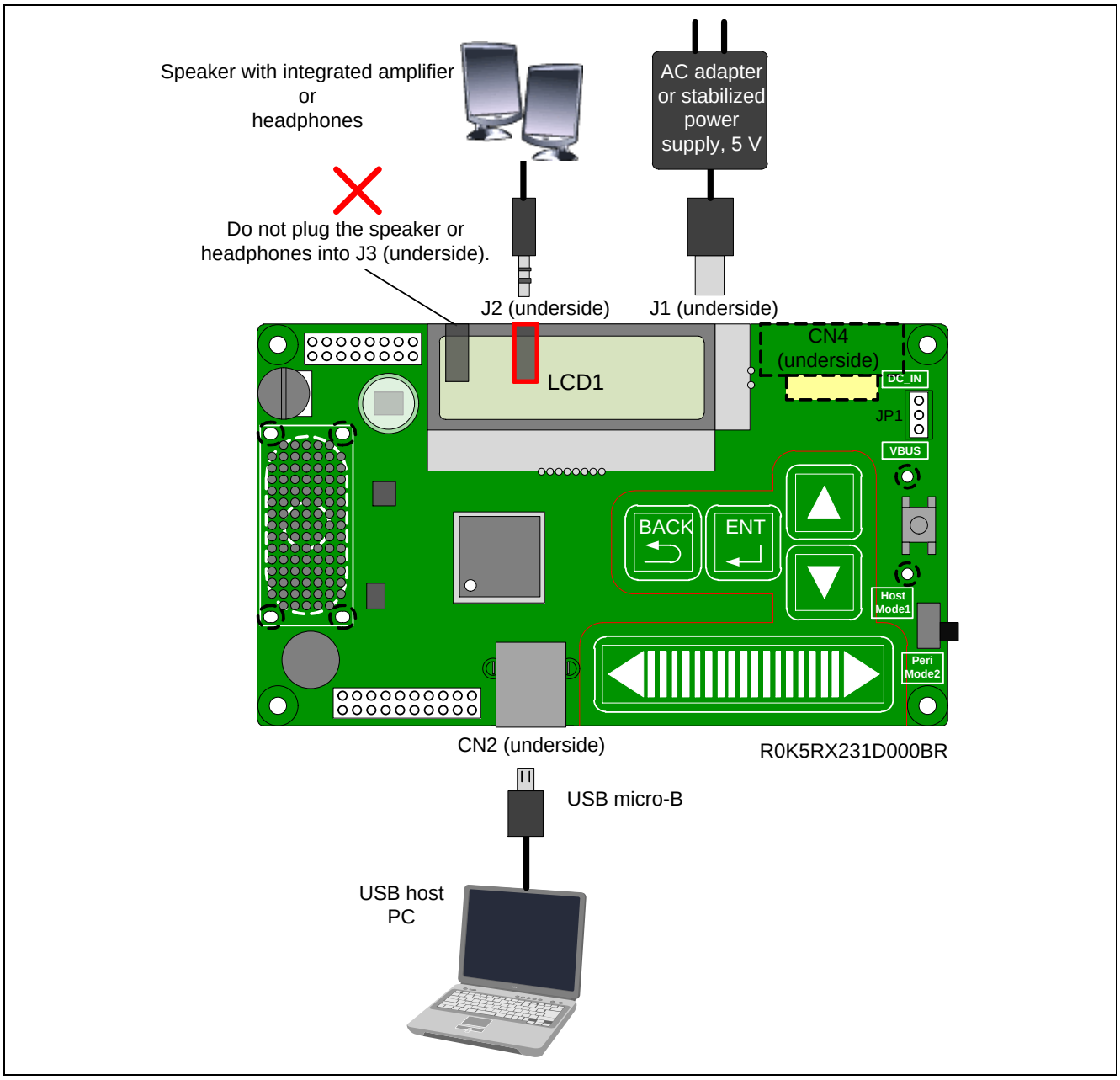


Figure 2.3 Connections of RX231 HMI Kit, USB host, and Speaker with Integrated Amplifier or Headphones (Self-Powered Operation)

3. Overview of Demo Software

This section describes the functions and structure of the software, the FIT modules used, and the file configuration of the project folder.

3.1 Functionality

The software provides the following functionality:

- USB communication with host
- Playback of sound sources sampled at 44.1 kHz, 16 bits, and 2 channels (stereo)
- Music play, stop, and pause
- Volume adjustment
- Mute setting

Note: No clock synchronization is performed.

3.2 Software Configuration

Figure 3.1 shows the software configuration of the software.

The software uses the functionality of the RX231 as a basis and uses Firmware Integration Technology (FIT) from Renesas to implement control. FIT APIs are used to control each driver, and the application (APL) uses the APIs provided by the FIT modules and drivers to operate.

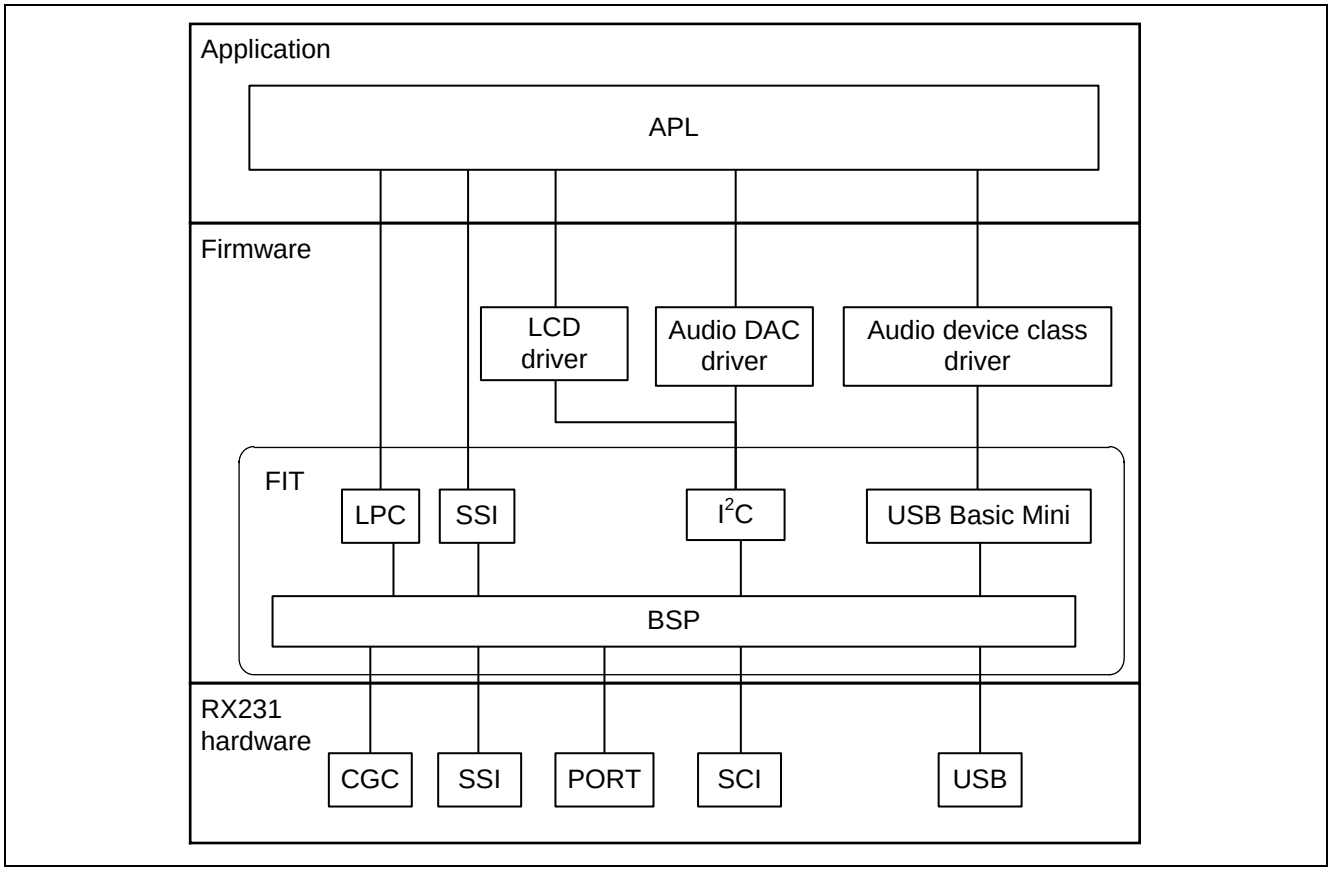


Figure 3.1 Software Configuration

Supplement:

Refer to the hardware manual (item No. 11 in Table 1.1, Related Documents) for details of the various functions of the RX231 hardware.

- BSP: RX Family board support package module with FIT
- LPC: LPC module with FIT
- SSI: SSI module with FIT
- I^2C: simple I^2C module with FIT
- USB Basic Mini: USB Basic Mini with FIT

The drivers are described in section 5, Audio Device Class Driver, and 4.6, Audio DAC Driver. The LCD driver is not described in this document as it is already covered in "RX231kit_free" RTK5RX2310P000F0ZR, the free version of the RX231 HMI Kit. For details of the free project, refer to item No. 13 in Table 1.1, Related Documents.

3.3 Address Space

The RX231 HMI Kit incorporates the R5F52318ADFP (RX231, ROM: 512 KB, RAM: 64 KB) as its MCU.

Figure 3.2 shows the address space used exclusively by the software.

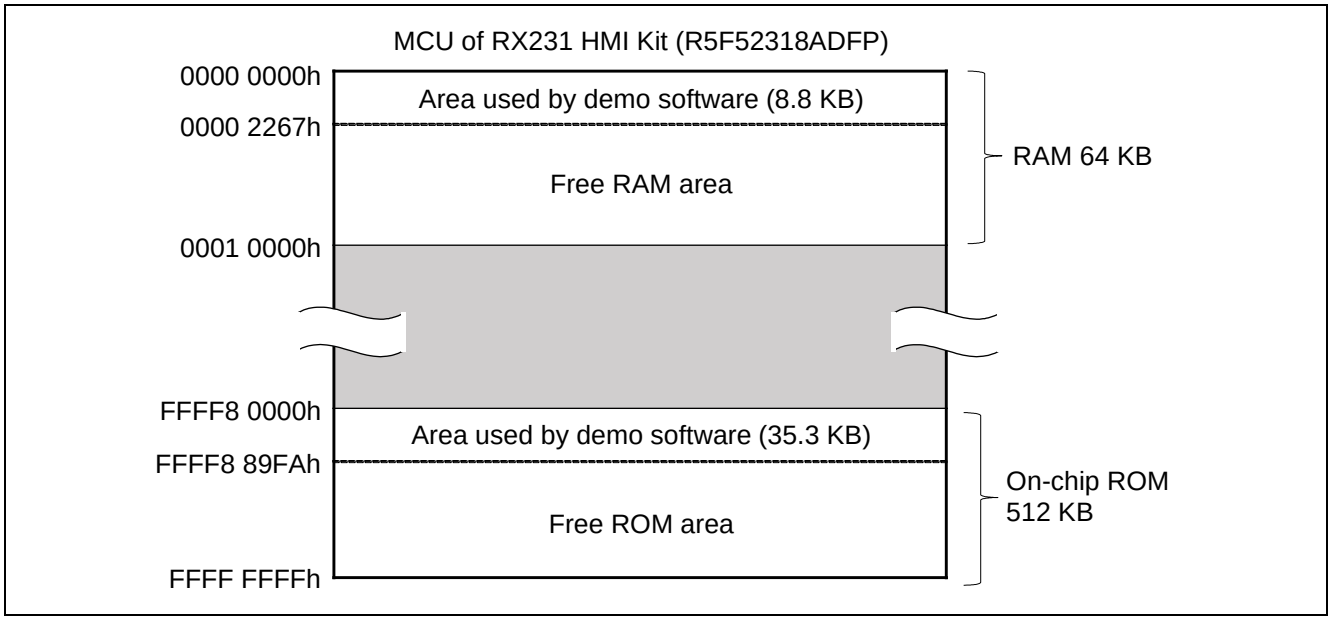


Figure 3.2 Address Space Used Exclusively by the Software

3.4 File Configuration

Table 3.1 shows the file configuration of the software. Note that the audio device class driver is indicated below as ADCD.

For details of the FIT folders, refer to items No. 6 to No. 10 in Table 1.1, Related Documents.

Table 3.1 File Configuration of the Software

Folder Name/File Name	Overview
RX231kit_paudio	—
.cproject	e² studio file
.HardwareDebuglinker	
.info	
.project	
RX231kit_Audio.rcpc	
demo_src	APL source folder
lcd.c	LCD control function source file
main.c	Main function source file
audio_apl.c	APL source file
audio_apl_descriptor.c	USB descriptor source file
inc	Application include folder
lcd.h	LCD control function include file
audio_apl.h	Application include file
r_dac	Audio DAC driver folder
r_dac_if.h	API function include file
readme.txt	—
src	Audio DAC driver source folder
r_dac_api.c	API function source file
r_dac_driver.c	driver function source file
inc	Audio DAC driver include folder
r_dac.h	Audio DAC driver function include file
r_usb_paudio	ADCD folder
r_usb_paudio_if.h	API function include file
readme.txt	—
ref	ADCD reference folder
r_usb_paudio_config_reference.h	ADCD reference file
src	ADCD source folder
r_usb_paudio_api.c	API function source file
r_usb_paudio_driver.c	Driver function source file
inc	ADCD include folder
r_usb_paudio.h	ADCD driver function include file
r_config	Configuration folder
r_usb_paudio_config.h	ADCD configuration file
r_bsp_config.h	BSP module FIT configuration file
r_lpc_rx_config.h	LPC module FIT configuration file
r_sci_iic_rx_config.h	Simple I²C module FIT configuration file
r_ssi_api_rx_config.h	SSI module FIT configuration file
r_usb_basic_mini_config.h	USB Basic Mini FIT configuration file
readme.txt	—

Folder Name/File Name	Overview
r_bsp	BSP module FIT folder
r_lpc_rx	LPC module FIT folder
r_sci_iic_rx	simple I^2C module FIT folder
r_ssi_api_rx	SSI module FIT folder
r_usb_basic_mini	USB Basic Mini FIT folder

3.5 FIT Modules Used

Table 3.2 lists the FIT modules used by the software. For details of each FIT module, refer to items No. 6 to No. 10 in Table 1.1, Related Documents. Note that the FIT modules used by the software are available for download on the Renesas website.

Table 3.2 FIT Modules Used by the Software

FIT Module	Revision	Folder Name	Scope of Use
Board support package module with FIT	3.30	r_bsp	All aspects of the software
USB Basic Mini FIT	1.02	r_usb_basic_mini	ADCD
SSI module FIT	1.20	r_ssi_api_rx	APL
Simple I^2C module FIT	1.60	r_sci_iic_rx	APL Audio DAC driver LCD driver
LPC module FIT	1.40	r_lpc_rx	APL

3.5.1 Revisions to USB Basic Mini with FIT

The functionality of USB Basic Mini with FIT is used to operate the ADCD, but some changes have been made. Bear this in mind when using the demo software as reference to develop your own software. Table 3.3 lists the changes.

Table 3.3 Location and Description of Changes to USB Basic Mini with FIT

Target File	Target Function	Description of Changes
r_usb_pdriver.c	usb_pstd_set_interface3()	This function contains no processing other than normal value determination, so a call has been added to a callback function to perform class request processing. (Changing and notification of the Alternate value is performed by the callback function.)
	usb_pstd_get_interface1()	Processing to return a fixed value (USB_0) has been removed and processing added to call a callback function to perform class request processing. (The Alternate value is returned by the callback function.)

4. Application

The application (APL) included with the software operates using APIs provided by the drivers and FIT modules. It implements the following functionality:

- Uses the SSI to transfer audio data received via USB to the PCM1774 audio DAC IC (audio DAC).
- Controls the audio DAC in response to mute or volume change indications from the USB host.
- Performs state transitions when USB Attach, Detach, Suspend, and Resume events are detected (including transition to the low-power state).
- Performs on/off switching of the LCD backlight to match LCD indications and state transitions.

The APL is composed of three parts: initial settings, main loop, and interrupt handler. These three types of processing are described below.

4.1 Initial Settings

The initial settings portion includes processing to clear event information, make initial settings for each driver, make FIT initial settings, show the initial indication on the LCD, and stop operation of unused ICs and modules.

The processing for making the sequence of initial settings is implemented in the function audio_apl_init().

4.2 Main Loop

The main loop portion of the software is divided into sections of processing for various events that can occur. Each event and the processing associated with it are described below.

4.2.1 Events

APL receives notifications from the ADCD and controls the various drivers in response. Each notification is managed as an event. The main loop is constantly monitoring for the occurrence of events.

(a) **Structure**

Events and their associated data are managed using the following structure provided by APL.

```
typedef struct                          /* Structure for event management */
{
    uint8_t     event[EVENT_MAX];       /* State for application */
    uint16_t    data[EVENT_MAX];        /* Event's data */
    uint16_t    event_cnt;              /* Event count */
} audio_eventinfo_t;
```

Variables for storing the event and data are provided, and information on an event and its data can be obtained by calling the function audio_event_get() with these variables as arguments.

Table 4.1 and Table 4.2 list the specifications of function audio_event_get and function audio_event_set, respectively.

Table 4.1 audio_event_get

Functionality	Gets information on the event that has occurred.		
Declaration	void audio_event_get(uint8_t* event, uint8_t* data)		
Arguments	uint8_t*	event	Event name storage destination address
	uint8_t*	data	Associated data storage destination address
Return values	—	—	
Specifications	Stores the event code in the variable event, which was passed as an argument, and the associated data in the variable data, also passed as an argument. If no event has occurred, APL_EV_NONE is stored in event. If there is no data, DATA_NONE is stored in data.		

Table 4.2 audio_event_set

Functionality	Stores information on an event		
Declaration	uint8_t audio_event_set(uint8_t event_name, uint16_t data)		
Arguments	uint8_t	event	Event name
	uint16_t	data	Data
Return values	uint8_t	AUDIO_EVENT_SET_ERROR: Event count exceeded error	
		AUDIO_EVENT_SET_SUCCESS: Normal processing end	
Specifications	Stores the event and data passed as arguments. The number of events that can be stored is specified as five in the initial settings (macro definition: EVENT_MAX).		

(b) **List of Events**

Table 4.3 lists the events defined by APL.

Table 4.3 Events and Associated Data

Event Name	Event Description	Data Name	Data Description
APL_EV_NONE	No event	DATA_NONE	No data
APL_EV_USB_STREAM	USB transfer start/stop notification	STREAM_PLAY	Isochronous transfer start
		STREAM_STOP	Isochronous transfer stop
APL_EV_USB_VOL	Volume change notification	data	Volume data specified by USB audio device class
APL_EV_USB_MUTE	Mute setting change notification	USB_MUTE_ON	Mute on
		USB_MUTE_OFF	Mute off
APL_EV_USB_RX_COMPLETE	USB receive complete notification	DATA_NONE	No data
APL_EV_USB_STS_CHANGE	USB transfer start/stop notification	USB_STS_DETACH	Next state code supplied by USB Basic Mini with FIT
		USB_STS_DEFAULT	
		USB_STS_ADDRESS	
		USB_STS_SUSPEND	
		USB_STS_RESUME	

Note: For details of the volume data, refer to the explanation under “Volume Control” in Universal Serial Bus Device Class Definition for Audio Devices, Rev. 1.0 (No. 2 in Table 1.1, Related Documents).

4.2.2 Operation Sequence

The main loop performs the following processing. Figure 4.1 illustrates the operation sequence of APL.

(a) Uses function audio_event_get() to get the event and associated data, at the beginning of the loop.
(b) If the event is APL_EV_USB_STREAM, identifies the contents of the data and, if STREAM_PLAY, issues a USB data receive request.
(c) If the event is APL_EV_USB_VOL, performs calculation to match the audio DAC specifications on the volume data transferred from the USB host, and uses simple I^2C communication to write the result to the audio DAC's digital volume register.
(d) If the event is APL_EV_USB_MUTE, first determines whether the data value is USB_MUTE_ON or USB_MUTE_OFF. Uses simple I^2C communication to write the data mute setting to the audio DAC's mute register.
(e) If the event is APL_EV_USB_COMPLETE, issues a USB data receive request and, if the SSI start condition is met, starts the SSI.
(f) If the event is APL_EV_USB_STS_CHANGE, performs state transition processing in accordance with the data state code. The state transition processing is implemented by the function audio_change_sts().

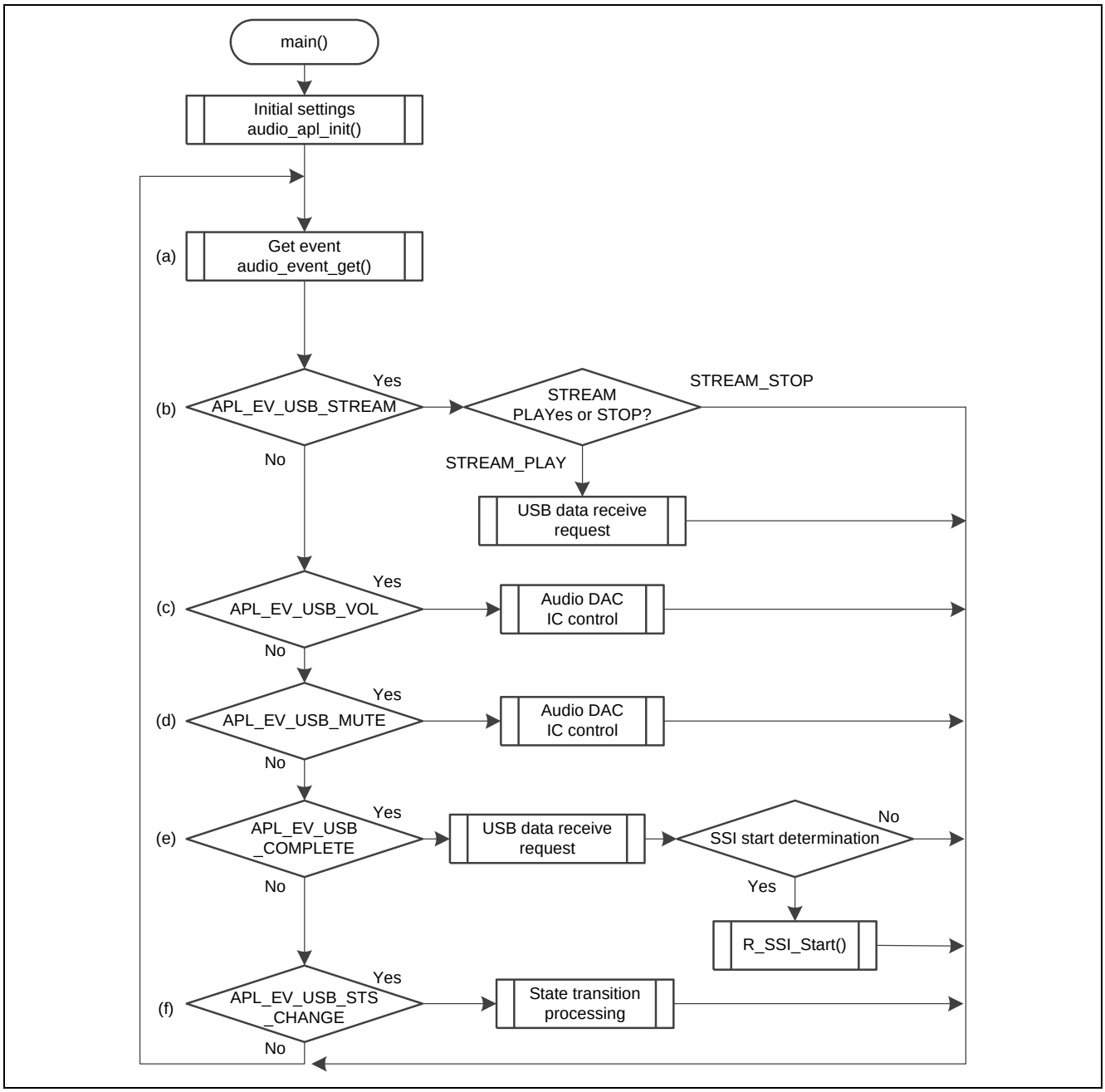


Figure 4.1 Main Loop Operation Sequence

4.3 Interrupts

Table 4.4 lists the interrupts used by APL.

Table 4.4 Interrupts Used by APL

Channel	Interrupt Source	Description
SSI0	SSITXI0 (transmit data empty interrupt)	Generated when the number of transmit data bytes stored in the SSI transmit FIFO is equal to or less than the value specified by the transmit FIFO threshold setting bits.

(a) **SSITXI0 Generation Timing**

The software makes the SSI FIT configuration settings shown in Table 4.5.

Table 4.5 Partial Listing of SSI FIT Configuration Definitions

Configuration Definition	Setting Value	Overview
SSI_CH0_DATA_WIDTH	16 u	PCM data width
SSI_CH0_TTRG_NUMBER	4 u	Value at which TDE flag is set

With these settings, the RX231's transmit FIFO threshold setting bits are set to 1h. In this case, the interrupt is generated when two stages of the 32-bit × 8-stage FIFO are empty.

Table 4.6 SSIFCR TTRG Setting

Bit Field	Initial Value	Setting Value
Transmit FIFO threshold setting bits (TTRG)	0h	1h

(b) **Processing Using SSITXI0**

SSITXI0 is enabled by the SSI transfer start processing. When this interrupt occurs, APL transfers 64 bits of data from the RAM buffer to the SSI FIFO.

When USB isochronous transfer stops and there is no data to be fetched from the RAM buffer, the software performs SSI transfer end processing. SSITXI0 is disabled at this point. From SSI transfer start to transfer end, APL continues to transfer data from the RAM buffer to the SSI FIFO.

Note: For details on interrupts, refer to the RX231 hardware manual (item No. 11 in Table 1.1, Related Documents) and the SSI FIT application note (item No. 8 in Table 1.1, Related Documents).

4.4 Low-Power State

When the software detects a Suspend instruction from the USB host, it transitions the RX231 HMI Kit to the low-power state. Table 4.7 lists the states of the MCU, audio DAC, and LCD.

Note that the low-power state is canceled when a Resume instruction from the USB host is detected.

Table 4.7 Low-Power State

USB State Transition	MCU (RX231)	Audio DAC	LCD
Suspend	Software standby mode	Low-power state	Backlight off

4.5 Audio Data

The software transfers audio data using the bit rate and number of channels shown in Table 4.8.

Table 4.8 Audio Data Information

Sampling Frequency	Bit Rate	Number of Channels
44.1 kHz	16 bits	2 (stereo)

These specifications require that 44.1 samples be transferred each millisecond.

44.1 [samples/msec.] × 16 [bits] × 2 [channels] = 176.4 [bytes/msec.]

Thus, 176.4 bytes of audio data must be transferred each millisecond.

But the minimum transfer size for USB data transfer is 1 byte. To get around this, the transfer rate is reconciled by repeatedly performing transfers of 176 bytes × 9 times + 180 bytes × 1 time.

The RAM buffer (see below) has a maximum packet size of 180 bytes, so 180 bytes is used as the size of one plane of data.

4.5.1 RAM Buffer

The RAM buffer used to store audio data is composed of three planes, each 180 bytes in size. The RAM buffer structure is shown below. This structure is provided by APL.

```
typedef struct                              /* Structure for PCM RAM buffer */
{
    uint8_t     data[PCM_BUF_NUM][PCM_BUF_SIZE];    /* PCM data */
    uint8_t     r_buf_num;          /* The buffer number to write USB data */
    uint8_t     w_buf_num;          /* The buffer number to write in SSI */
    uint16_t    w_pos;              /* SSI write pointer */
    uint16_t    r_len[PCM_BUF_NUM]; /* The length of stored data in each
                                       buffer */
} audio_buf_t;
```

4.5.2 Flow of Audio Data

The flow of audio data is described below. Figure 4.2 illustrates the entire flow sequence.

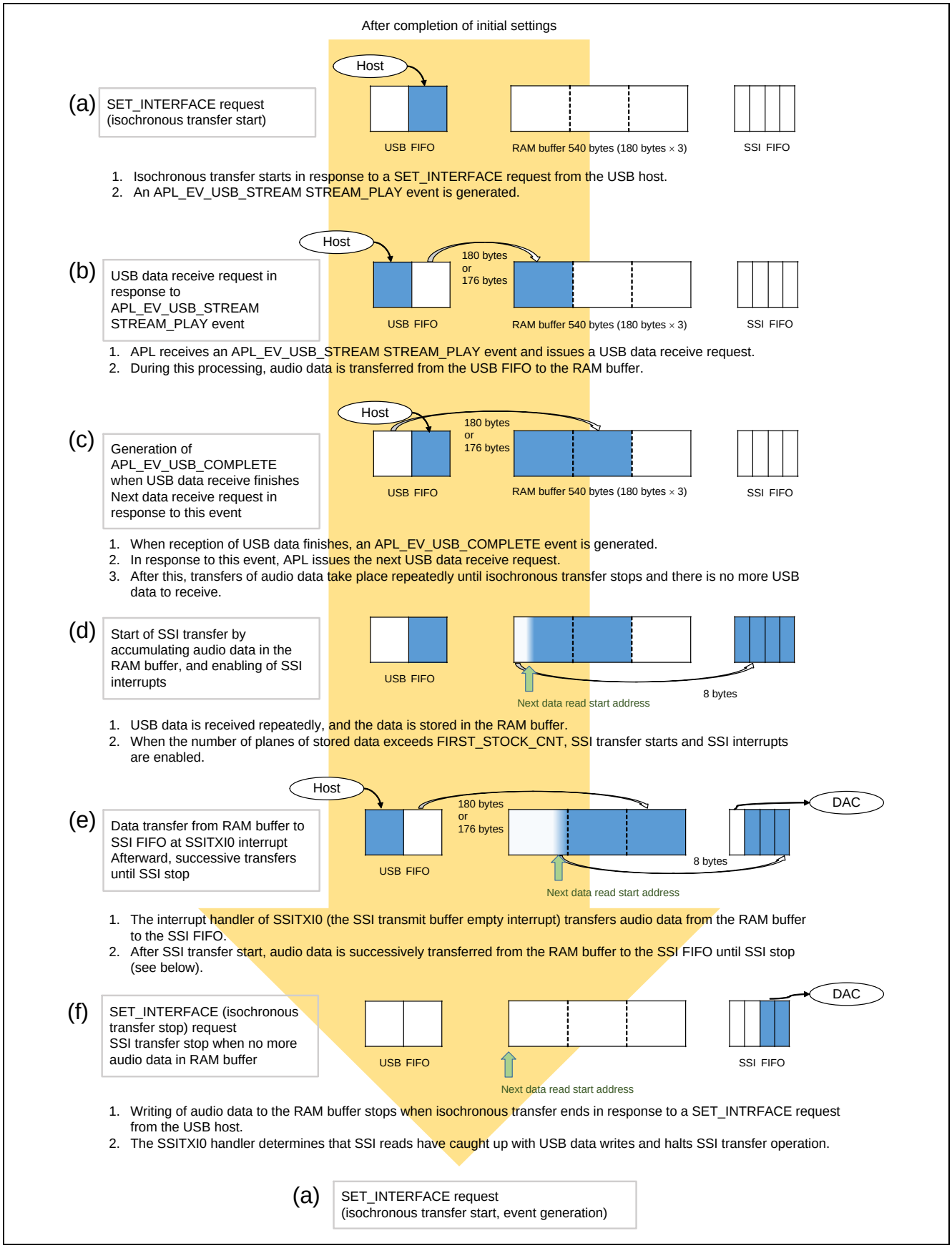


Figure 4.2 Flow of Audio Data Controlled by the Software

4.5.3 Alterable Settings

The buffer plane count and initial storage count at SSI transfer start can be changed. These definitions are contained in audio_apl.h.

Table 4.9 APL Alterable Settings

Macro Name	Initial Value	Description
PCM_BUF_NUM	3	RAM buffer plane count
FIRST_STOCK_CNT	2	SSI transfer start condition SSI transfer starts when the number of bytes of data in the RAM buffer exceeds this value.

4.6 Audio DAC Driver

The audio DAC driver controls the PCM1774 audio DAC, manufactured by Texas Instruments, that is mounted on the board of the RX231 HMI Kit.

The driver uses the simple I^2C module with FIT. Before using the driver, apply initial settings to the simple I^2C module.

4.6.1 Basic Functionality

The audio DAC driver performs the following processing

- Audio DAC power-on and initial settings
- Audio DAC power-off
- Audio DAC mute setting/volume change
- Audio DAC transition to low-power state
- Audio DAC cancellation of low-power state

4.6.2 Header File

All API calls and the interface definitions they support are contained in r_dac_if.h.

4.6.3 API Functions

The functionality of the API functions is described below. When using a different audio DAC, change the processing of these functions as necessary.

Table 4.10 lists the API functions of the audio DAC driver and their functionality.

Table 4.10 API Functions of Audio DAC Driver

API Function	Description
void R_DAC_Open(void)	Audio DAC power-on and initial settings
void R_DAC_Close(void)	Audio DAC power-off
uint8_t R_DAC_Control(uint8_t param_type, uint16_t data)	Mute setting/volume change
void R_DAC_Suspend(void)	Transition to low-power state
void R_DAC_Resume(void)	Cancellation of low-power state

(a) **R_DAC_Open**

Audio DAC power-on and initial settings

Format

void R_DAC_Open(void)

Arguments

— —

Return values

— —

Description

Powers on the audio DAC and applies initial settings.

This function performs the following processing:

1. Powers on the SG-210 oscillator that supplies the SSI communication clock.
2. Initializes and powers on the audio DAC.

Supplement

Apply initial settings to the simple I^2C module with FIT before calling this function.

Run this function before calling other audio DAC driver functions.

Example

```
void sample_main( void )
{
     /* I2C module initialize */
     sample_iic_init();

     /* power ON Audio DAC */
     R_DAC_Open();

     while( 1 )
     {
          /* main loop process */
     }
}
```

(b) **R_DAC_Close**

Audio DAC power-off

Format

void R_DAC_Close(void)

Arguments

— —

Return values

— —

Description

Powers off the audio DAC.

This function performs the following processing:

1. Powers off the audio DAC.
2. Powers off the oscillator (SG-210) that supplies the SSI communication clock.

Supplement

—

Example

```
void sample_task( void )
{
          :
    R_DAC_Close();     /* power OFF Audio DAC */
}
```

(c) R_DAC_Control

Audio DAC mute setting/volume change

Format

uint8_t R_DAC_Control(uint8_t param_type, uint16_t data)

Arguments

param_type	Code of parameter to be changed
data	Setting value

Return values

DAC_MUTE_PARAM_ERROR	Illegal mute setting
DAC_VOLUME_PARAM_ERROR	Illegal volume setting value
DAC_PARAM_ERROR	Illegal parameter code
DAC_CONTROL_SUCCESS	Success

Description

Overwrites the setting value specified by an argument as the parameter to be changed.

Pass a value to the first argument param_type and a setting value to the second argument data as shown below.

param_type	DAC_MUTE_SET	Change mute setting.
data	DAC_MUTE_ON	Turn mute on.
	DAC_MUTE_OFF	Turn mute off.
param_type	DAC_VOL_SET	Change volume setting.
data		Volume setting value (USB data)

The function uses simple I^2C communication to write to the audio DAC register the parameter code determined from the first argument and the setting value from the second argument.

Supplement

—

Sample code of a usage example is shown on the next page.

Example

```
void sample_main( void )
{
    uint8_t    event;
    uint16_t   data;
    uint8_t    err;

    /* I2C module initialize */
    sample_iic_init();

    /* power ON Audio DAC */
    R_DAC_Open();

        :
    while( 1 )
    {
        /* Get an event and related data */
        audio_event_get( event, data );

        switch (event)
        {
            case APL_EV_USB_VOL:
                /* Set the Audio DAC volume register */
                R_DAC_Control( DAC_VOLUME_SET, data );
                if ( err != DAC_CONTROL_SUCCESS )
                {
                    /* error process */
                }
            break;
            case APL_EV_USB_MUTE:
                if (USB_MUTE_ON == data)
                {
                    R_DAC_Control( DAC_MUTE_SET, USB_MUTE_ON );
                    if ( err != DAC_CONTROL_SUCCESS )
                    {
                        /* error process */
                    }
                }
                else if (USB_MUTE_OFF == data)
                {
                    R_DAC_Control( DAC_MUTE_SET, USB_MUTE_OFF );
                    if ( err != DAC_CONTROL_SUCCESS )
                    {
                        /* error process */
                    }
                }
            break;
            default:
                    /* No Action */
            break;
        }
        /* Other process */
            :
    }
}
```

(d) R_DAC_Suspend

Audio DAC transition to low-power state

Format

void R_DAC_Suspend(void)

Arguments

— —

Return values

— —

Description

Transitions the audio DAC to the low-power state.

This function performs the following processing:

1. Powers off some of the modules of the audio DAC.
2. Powers off the oscillator (SG-210) that supplies the SSI communication clock.

Supplement

—

Example

```
void sample_task( void )
{
          :
    R_DAC_Suspend();     /* Standby Audio DAC */
}
```

(e) **R_DAC_Resume**

Audio DAC return from low-power state

Format

void R_DAC_Resume (void)

Arguments

— —

Return values

— —

Description

Returns the audio DAC to the normal state after it was put into the low-power state by R_DAC_Suspend().

This function performs the following processing:

1. Powers on the oscillator (SG-210) that supplies the SSI communication clock.
2. Powers on the modules turned off by R_DAC_Suspend().

Supplement

—

Example

```
void sample_task( void )
{
          :
    R_DAC_Resume();     /* Resume Audio DAC */
}
```

4.7 Descriptors

The software's descriptor information is contained in audio_apl_descriptor.c. In addition, the audio play, stop, mute, and volume change functionality included in USB audio device class 1.0 is supported.

Make sure to use the customer's own numbers for Product ID and Vendor ID.

Table 4.11 lists the descriptors provided by the software.

Table 4.11 Overview of Descriptors

Descriptor	Description
Device descriptor	Device-specific information
Configuration descriptor	Descriptor related to device configuration
Audio control interface descriptor	Descriptor related to functionality provided by the device
Standard AC interface descriptor	USB 2.0-compliant interface information
Class-specific AC interface header descriptor	Descriptor related to the audio device class version (1.0) and succeeding class-specific, terminal, and unit information
Input terminal descriptor	Descriptor indicating USB streaming of data input by host
Output terminal descriptor	Descriptor related to the device type that is conveyed to the host
Audio control feature unit descriptor	Descriptor related to elements controllable by the device
Audio streaming interface descriptor	Descriptor related to audio data receive functionality
Standard AS interface descriptor (for alternate setting 0)	Descriptor related to interface for audio output stop
Standard AS interface descriptor (for alternate setting 1)	Descriptor related to interface for data stream
Class-specific AS interface descriptor	Descriptor related to linkage between input terminal and data format type
Class-specific AS format type descriptor	Descriptor related to format of communication data
Standard AS isochronous audio data endpoint descriptor	USB 2.0-compliant endpoint information
Class-specific AS isochronous audio data endpoint descriptor	Descriptor related to endpoint for data stream
String descriptor	Specifies character information (company name, etc.)

Table 4.12 lists the relationships between the descriptors and variables of the software.

Table 4.12 Relationships between Descriptors and Variables of the Software

Descriptor Name	Type	Variable Name
Device descriptor	uint8_t	g_audio_device_descriptor []
Configuration descriptor	uint8_t	g_audio_configuration []
Standard AC interface descriptor		
Class-specific AC interface header descriptor		
Input terminal descriptor		
Output terminal descriptor		
Audio control feature unit descriptor		
Standard AS interface descriptor (for alternate setting 0)		
Standard AS interface descriptor (for alternate setting 1)		
Class-specific AS interface descriptor		
Class-specific AS format type descriptor		
Standard AS isochronous audio data endpoint descriptor		
Class-specific AS isochronous audio data endpoint descriptor		
String descriptor	uint8_t	*g_audio_str_ptr[]*[1]
		g_audio_string_descriptor1[]
		g_audio_string_descriptor2[]
		g_audio_string_descriptor3[]
		g_audio_string_descriptor4[]
		g_audio_string_descriptor5[]

Note 1. *g_audio_str_ptr[] is an array containing g_audio_string_descriptor1[] to g_audio_string_descriptor5[].

4.8 User Callback Function

APL must provide a callback function in order to use the ADCD API function R_USB_PaudioReceiveData().

The callback function is passed as an argument a USB communication structure of type usb_utr_t, which contains the transmit/receive remaining data length, status, and transmission completed information.

The software uses the callback function cb_audio_receive_complete() to manage the RAM buffer and store USB reception completed events. The receive data length contained in the data communication structure is used for RAM buffer management.

The function R_USB_PaudioReceiveData() uses the R_usb_pstd_TransferStart(), an API function of USB Basic Mini with FIT. For a description of the processing and callback function details, refer to item No. 7 in Table 1.1, Related Documents.

5. Audio Device Class Driver

The ADCD is a driver that, in combination with the USB Basic Mini with FIT module, enables the RX231 HMI Kit to operate as a USB audio device class 1.0-compliant device.

Calling the functions for ADCD initial settings, registration, and main loop processing causes the USB Basic Mini with FIT module to be launched automatically. APL does not need to perform processing related to the USB Basic Mini with FIT module.

The basic functionality, configuration definitions, and API functions of the ADCD are described below.

5.1 Basic Functionality

The ADCD provides the following functionality:

- Transfer of audio data using isochronous out transfer
- Transfer of audio parameters (mute setting and volume setting) using control transfer
- Control transfer receive end notification (callback) to APL
- USB state transition notification (callback) to APL

Notifications are described in detail in 5.3, Class Driver Registration.

5.2 Class Requests

Table 5.1 lists the class requests supported by the ADCD.

Table 5.1 Class Requests Supported by ADCD

Class Request	Code	Description
GET_CUR	0x81	Returns the current value of the audio parameter specified by the control selector.
SET_CUR	0x01	Changes the current value of the audio parameter specified by the control selector.
GET_MIN	0x82	Returns the minimum value of the audio parameter specified by the control selector.
GET_MAX	0x83	Returns the maximum value of the audio parameter specified by the control selector.
GET_RES	0x84	Returns the resolution of the audio parameter specified by the control selector.

Note: Standard requests (with the exception of GET_INTERFACE and SET_INTERFACE) are processed by the USB Basic Mini with FIT module. Refer to item No. 7 in Table 1.1, Related Documents. Note that the software uses a partially modified version of the USB Basic Mini with FIT module. For details, see 3.5.1, Revisions to USB Basic Mini with FIT.

USB audio device class 1.0 has units called “feature units” that control the basic functionality of each channel. The controlled functionality is selected by control selectors. Finally, linkages to channels, enabled functionality, etc., are specified by descriptors.

The ADCD has a single feature unit, and the supported control selectors are listed in Table 5.2.

Table 5.2 Functionality Supported by ADCD Using Control Selectors

Control Selector	Code	Description
MUTE_CONTROL	0x0100	Changes the mute setting.
VOLUME_CONTROL	0x0200	Changes the volume.

5.3 Class Driver Registration

In order to use the ADCD, it is necessary to register descriptors and callback functions.

Registration is accomplished by calling the function R_USB_PaudioRegistration() after calling the initial settings function R_USB_PaudioOpen(). The functions are described in detail in 5.5, API Function Specifications.

The structure usb_paudio_reg_t, shown in Table 5.3, is used to register descriptors and callback functions.

Table 5.3 usb_paudio_reg_t

Type		**Overview**
usb_paudio_reg_t		Structure for descriptor and callback function registration
Type	Member	Overview
uint16_t *	pipetbl	Register in this member the address of the pipe information table.
uint8_t *	devicetbl	Register in this member the address of the device descriptor table.
uint8_t *	configtbl	Register in this member the address of the configuration descriptor table.
uint8_t **	stringtbl	Register in this member the address of the string descriptor table.
usb_cbinfo_t	statediagram	Register in this member the callback function activated when a USB state transition occurs.
usb_paudio_cb_t	ctrlRxCB	Register in this member the callback function called when an audio control transfer is received. For details, see 5.3.1, Callback Functions.

Note: The structure usb_cbinfo_t is provided by the USB Basic Mini with FIT module. For details, refer to item No. 7 in Table 1.1, Related Documents.

5.3.1 Callback Functions

When a SET_INTERFACE or SET_CUR (MUTE_CONTROL or VOLUME_CONTROL) command is sent by the USB host, the ADCD calls the associated registered callback function. Callback functions are called by using the types listed in Table 5.4, and events and associated data are passed as arguments.

Table 5.4 (*usb_paudio_cb_t)(uint8_t, uint16_t)

Type			**Overview**
(*usb_paudio_cb_t)(uint8_t, uint16_t)			Callback function type used for control notifications
Arguments	Type	Value	Overview
	uint8_t	USB_PAUDIO_STREAM	Audio output setting
		USB_PAUDIO_MUTE	Mute setting
		USB_PAUDIO_VOLUME	Volume setting
	uint16_t	STREAM_PLAY	Audio output start
		STREAM_STOP	Audio output stop
		MUTE_ON	Mute on
		MUTE_OFF	Mute off
		Volume setting value	Volume setting value
Return values	—	—	

Table 5.5 shows the relationship between the timing of the call to the callback function and the arguments that are passed.

Table 5.5 Relationship between Callback Function Timing and Arguments

1st Argument (uint8_t)	**2nd Argument (uint16_t)**	**Issue Timing**
USB_PAUDIO_STREAM	STREAM_PLAY	When SET_INTERFACE Alternate = 1 request received
	STREAM_STOP	When SET_INTERFACE Alternate = 0 request received
USB_PAUDIO_MUTE	MUTE_ON	When SET_CUR MUTE_CONTROL Parameter Block = 1 request received
	MUTE_OFF	When SET_CUR MUTE_CONTROL Parameter Block = 0 request received
USB_PAUDIO_VOLUME	Volume setting value	When SET_CUR VOLUME_CONTROL request received

5.4 API Information

The ADCD API conforms to the Renesas API instruction standard.

5.4.1 Hardware Requirements

The operation of the ADCD has been confirmed on the following hardware.

R0K5RX231D000BR (RX231 HMI Solution Kit)

5.4.2 Header File

API calls and the interface definitions they support are contained in r_usb_paudio_if.h.

5.4.3 Configuration

Settings for volume setting and the pipe used in the driver are contained in r_usb_paudio_config.h. Table 5.6 lists the configuration definitions.

Table 5.6 ADCD Configuration Definitions

Macro Name	Initial Value	Overview
USB_CFG_PAUDIO_PIPE_OUT	USB_PIPE1 (0x0001)	Pipe number of isochronous transfer out pipe Select from the following: USB_PIPE1 (0x0001) USB_PIPE2 (0x0002)
USB_CFG_PAUDIO_VOL_MAX	0x0000	Volume maximum value
USB_CFG_PAUDIO_VOL_MIN	0xC100	Volume minimum value
USB_CFG_PAUDIO_VOL_RES	0x0100	Volume resolution

The USB functionality of the RX231 supports isochronous transfer on USB_PIPE1 and USB_PIPE2.

The initial values used in the software match the specifications of the audio DAC of the RX231 HMI Kit (digital volume change range: 0 dB to −63 dB).

5.5 API Function Specifications

The API functions are described in detail below.

Table 5.7 lists the API functions of the ADCD.

Table 5.7 API Functions of ADCD

API Function	Description
void R_USB_PaudioOpen(void)	ADCD initial settings
void R_USB_PaudioRegistration(usb_paudio_reg_t *paudio_reg)	Table and callback registration
void R_USB_PaudioReceiveData(uint8_t *Table, usb_leng_t size, usb_cb_t complete)	USB data receive request
void R_USB_PaudioDriver(void)	USB Basic Mini with FIT schedule management and task processing

5.5.1 R_USB_PaudioOpen

ADCD startup function

Format

void R_USB_PaudioOpen(void)

Arguments

— —

Return values

— —

Description

Launches the ADCD. Includes startup processing for the USB Basic Mini with FIT module.

Run this function before calling other ADCD functions.

This function performs the following processing:

1. Makes USB pin settings, USB module initialization, etc.
2. Calls R_USB_Open().
3. Calls R_usb_pstd_PcdOpen().

Supplement

—

Example

```
void sample_main(void)
{
    /* Audio Device Class driver initializing and registration */
    R_USB_PaudioOpen();
    sample_usb_registraion();

    while( 1 )
    {
        /* Main loop process */
    }
}
```

5.5.2 R_USB_PaudioRegistration

ADCD registration

Format

void R_USB_PaudioRegistration (usb_paudio_reg_t *paudio_reg)

Arguments

*paudio_reg Address of structure for registration

Return values

— —

Description

Registers descriptors and callback functions.

Run this function immediately after R_USB_PaudioOpen().

This function performs the following processing:

1. Calls R_usb_pstd_DriverRegistration() to register the following in the USB Basic Mini with FIT module:
   - Pipe information table
   - Device descriptor table
   - Configuration descriptor table
   - String descriptor table
   - Callback function that runs when USB state transition occurs
   - ADCD control transfer processing function
2. Registers the callback function that is called when an audio control transfer is received.

Supplement

- For the information that is registered, see Table 5.3, usb_paudio_reg_t, in 5.3, Class Driver Registration.
- R_usb_pstd_DriverRegistration() is an API function of the USB Basic Mini with FIT module. For details, refer to item No. 7 in Table 1.1, Related Documents.

Example

The code of the software's APL function audio_usb_registration() is shown below as an example.

```
void audio_usb_registration( void )
{
    usb_paudio_reg_t    audio_reg;

    /* USB endpoint Table */
    audio_reg.pipetbl      =    &g_audio_ep_tbl[0];
    /* Device descriptor */
    audio_reg.devicetbl    =    &g_audio_device_descriptor[0];
    /* Configuration Descriptor */
    audio_reg.configtbl    =    &g_audio_configuration[0];
    /* String Descriptor */
    audio_reg.stringtbl    =    (uint8_t**)&g_audio_str_ptr[0];
    /* USB state transition callback */
    audio_reg.statediagram =    cb_audio_change_device_state;
    /* USB control transfer complete callback */
    audio_reg.ctrlRxCB     =    cb_audio_usb_control_complete;

    /* Registration */
    R_USB_PaudioRegistration( &audio_reg );

}  /* eof audio_usb_registration() */
```

5.5.3 R_USB_PaudioReceiveData

USB data receive request

Format

void R_USB_PaudioReceiveData(uint8_t *Table, usb_leng_t size, usb_cb_t complete)

Arguments

*Table	Receive data storage buffer address
size	Number of bytes of data to be received
complete	Data receive end callback function address

Return values

— —

Description

Sends a USB data receive request to the USB Basic Mini with FIT module.

Calls the callback function specified by the third argument at receive end. For information on the receive end callback function, see 4.8, User Callback Function.

Supplement

—

Example

```
static audio_buf_t      g_audio_buf;
#define PCM_BUF_SIZE   (180)
void cb_sample_receive_complete( usb_utr_t * mess )


void sample_usb_receive( void )
{
    /* Receive USB data */
    R_USB_PaudioReceiveData( &g_audio_buf.data[0][0], PCM_BUF_SIZE,
    &cb_sample_receive_complete );
}

void cb_sample_receive_complete( usb_utr_t * mess )
{
    /* callback process */
}
```

5.5.4 R_USB_PaudioDriver

USB Basic Mini with FIT schedule management and task processing

Format

void R_USB_PaudioDriver(void)

Arguments

— —

Return values

— —

Description

Calls R_USB_cstd_Scheduler() to check for a task message.

If there is a message, calls R_usb_pstd_PcdTask(), an API function of the USB Basic Mini with FIT.

Supplement

This function should be called repeatedly from the main loop.

Example

```
void sample_main( void )
{
    /* Initialize Audio Device Class driver and registration */
    R_USB_PaudioOpen();
    sample_usb_registration();

    while( 1 )
    {
        /* main loop process */
            :
        R_USB_PaudioDriver();
    }
}
```

6. Development Environment

The software was developed, and its operation verified, in the environment outlined below. For instructions on using tools, refer to the manual of each tool.

Evaluation board

R0K5RX231D000BR (RX231 HMI Solution Kit)

Tools

a) e² studio integrated development environment, ver. 5.3.0.0.023, from Renesas Electronics
b) C/C++ Compiler Package for RX Family, ver. 2.05.00, from Renesas Electronics
c) E1 emulator from Renesas Electronics

Other

a) USB host PC (Microsoft Windows® 7, Windows® 8.1, or Windows® 10)
   Operation is supported by Windows standard drivers.
b) USB micro-B cable
c) Speaker with integrated amplifier or headphones

(a) **Connection to E1**

Connecting the E1 emulator makes it possible to overwrite and debug programs on the RX231. Follow the steps below to connect the E1 emulator.

1. Make sure the RX231 HMI Kit is powered off.
2. Connect the E1 cable to CN4 on the RX231 HMI Kit.
   Make sure to confirm the position of the “incorrect insertion prevention key” on the E1 cable.

For instructions on using the E1, refer to the user’s manuals of the E1 and the development environment.

To supply power from the E1, turn off the power supply (DC jack or USB) of the RX231 HMI Kit. Remove JP1 to cut off power input from both the DC jack and USB to ensure safe usage.

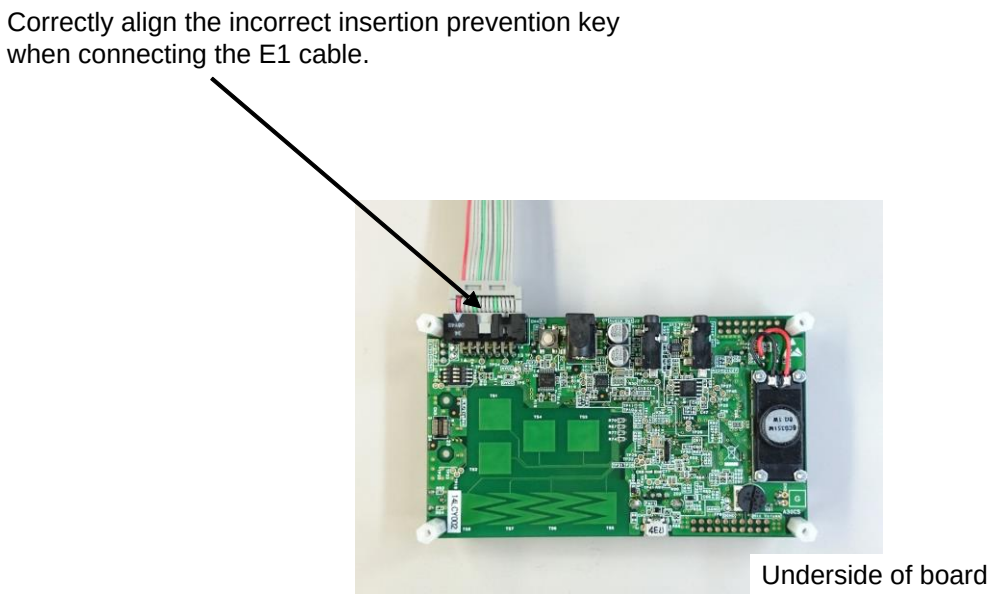


Figure 6.1 E1 Connection Diagram

(b) **Using an e² studio Project on CS+**

The software was created in the e² studio integrated environment. To use the software on CS+, import it by following the steps below.

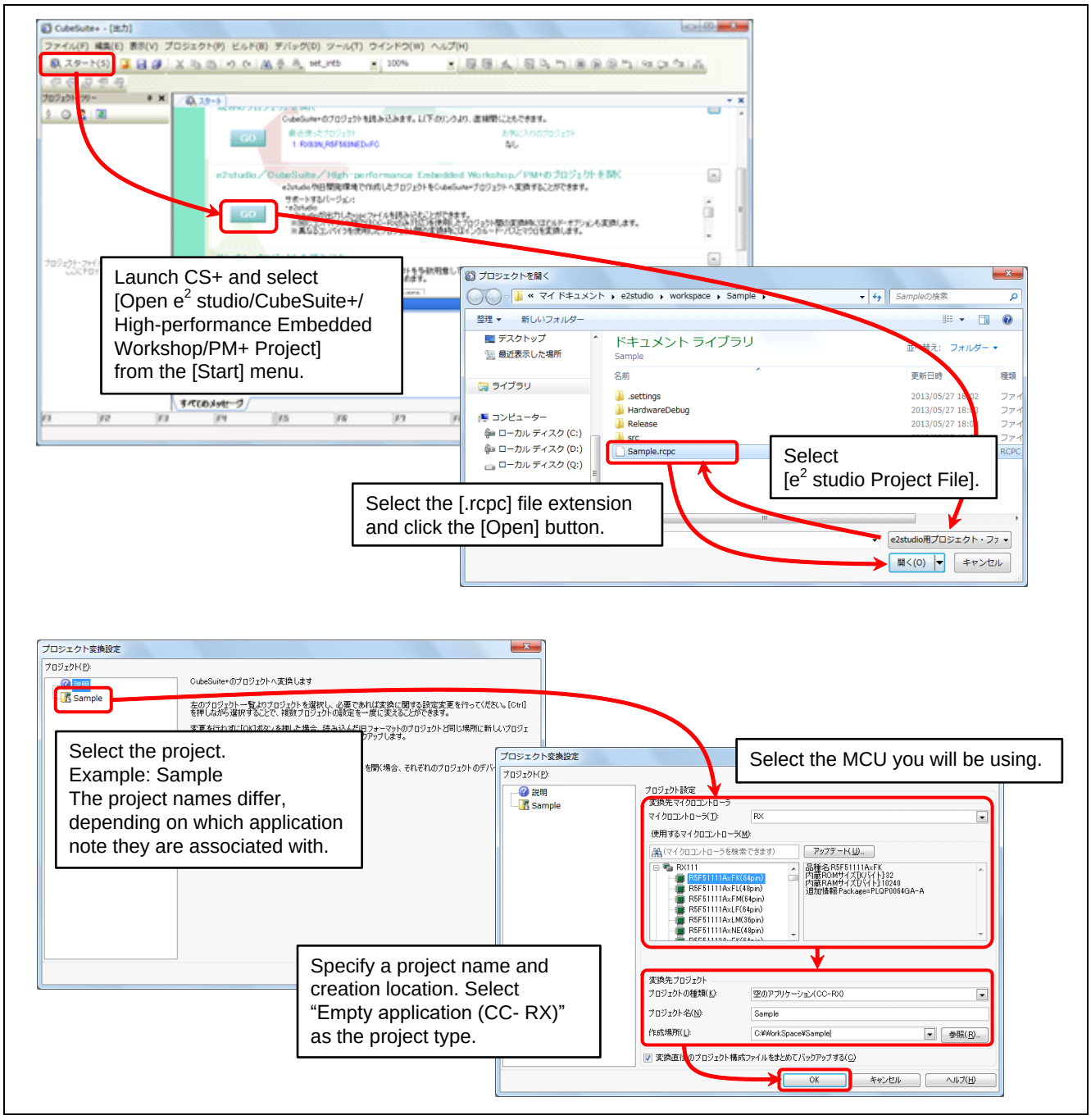


Figure 6.2 Reading an e² studio Project into CS+

7. Additional Notes

7.1.1 Processing of Unused Pins

Refer to RX231 Group User's Manual: Hardware, item No. 11 in Table 1.1, Related Documents.

7.2 Changing the USB ID

You cannot use the Vendor ID and Product ID values set in the software for your own products. Make sure to obtain a Vendor ID from USB-IF.

Note: USB-IF <http://www.usb.org/home>

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Apr. 26, 2017	—	First edition issued

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. 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the 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 disputes 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, drawing, chart, program, algorithm, 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 not alter, modify, copy, or otherwise misappropriate 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, copy or otherwise misappropriation of Renesas Electronics products.
5. 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; and 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.

   Renesas Electronics products are neither intended nor 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 damages (space and 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 third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the 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 radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond such specified ranges.
7. 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. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of 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 by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use 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.
9. 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 not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.3.0-1 November 2016)

RENESAS

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 6.0