Quick Setup Guide for VOICE-RA4E1 VUI Solution Kit

Renesas Advanced (RA) Family – RA4 Series

Description

Welcome to a Quick Setup Guide for VOICE-RA4E1 VUI Solution Kit. This guide will walk you through the setup required to exercise various features on the board, including all microphone inputs, speaker output and UART-to-USB communication. When migrating an application developed for another variant of the VOICE kit, cheat sheet in the final section can be used to quickly reconfigure the project for the new hardware target.

Objectives • Create a project for VOICE-RA4E1 • Enable serial terminal output • Enable digital microphones • Enable analog microphones • Enable audio playback	Prerequisites • Renesas VOICE-RA4E1 VUI Solution Kit • Renesas Flexible Software Package platform installation, which includes: • e^2 studio 2022-04 or newer • FSP 3.7.0 or newer • GCC Arm Embedded 10.3 (2021.10) or newer • PC running Windows 10 64-bit or newer with at least one USB port.
Skill Level • Basic familiarity with embedded electronics • Basic understanding of C language • Understanding of how to import projects into e^2 studio (optional- for use with ready checkpoint projects).	Time • 30 minutes for each section

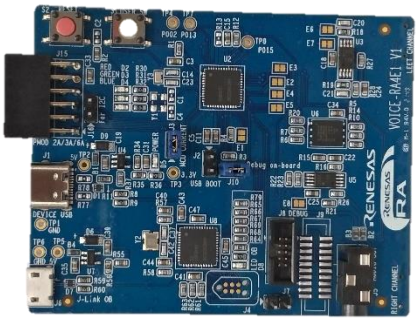
Quick Setup Guide Sections

0 Setting up the hardware

Overview

Before getting started, use the steps listed below to verify that your hardware can work correctly with your PC.

Procedural Steps

0.1	To begin working with VOICE-RA4E1 platform you will need the following:  VOICE-RA4E1 VUI Solution Kit USB micro-B cable (included with the kit)
0.2	Connect the VOICE-RA4E1 kit to the PC using USB micro-B cable and J-Link OB USB port (J6) in the bottom left corner of the board.
0.3	Verify that: • The blue LED (POWER, D5) near the centre of the board is on. • The green LED (LED OB, D8) near the bottom edge of the board is on and not flashing.
0.4	Your kit and operating environment are now set-up and ready for evaluation and development.

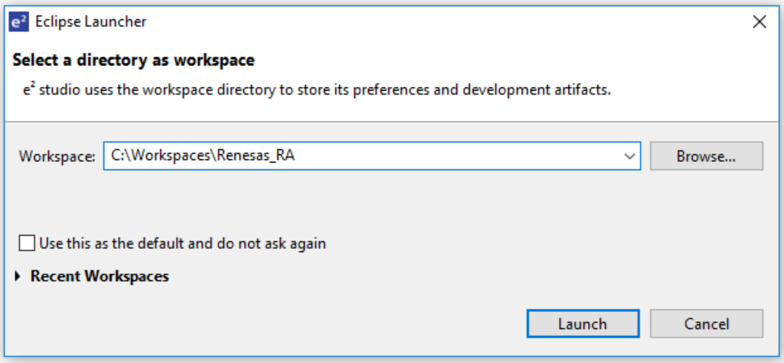
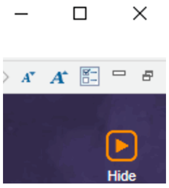
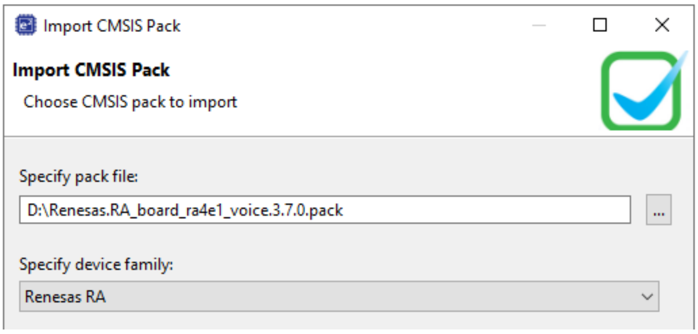
END OF SECTION

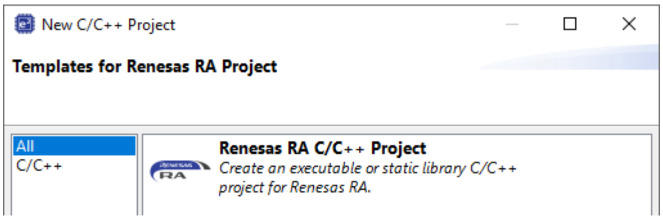
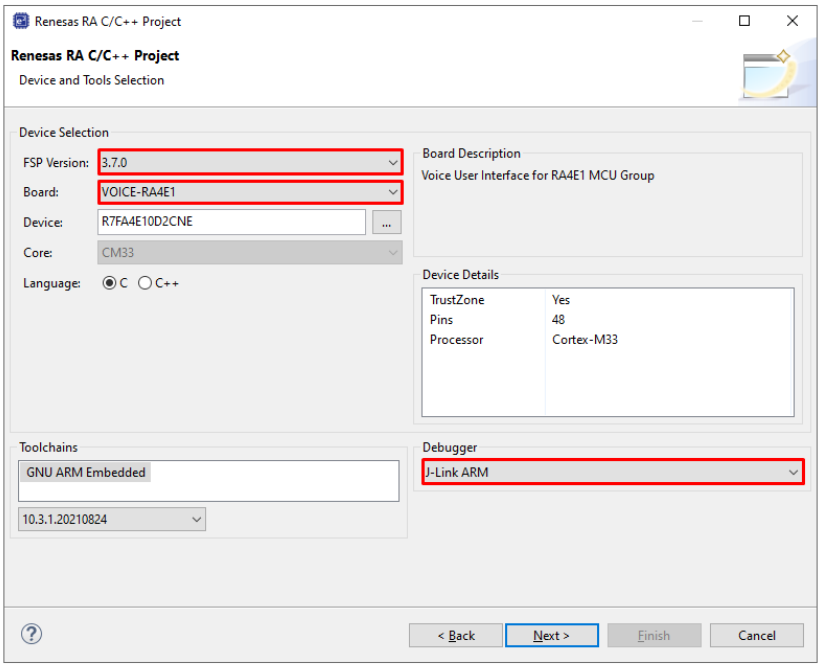
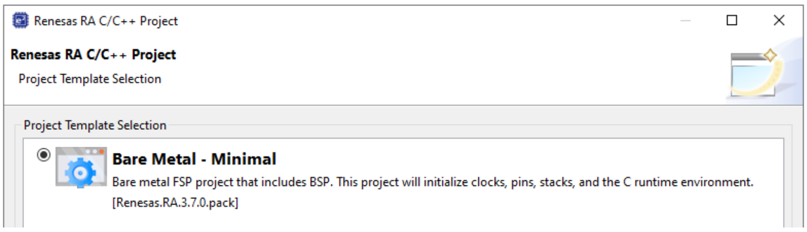
1 Installing BSP and creating an FSP project

Overview

Following section describes in details steps required to create an e^2 studio workspace and set up a project for RA4E1-VOICE kit.

Procedural Steps

1.1	Launch e^2 studio. e^2 studio can be launched from the Windows start menu or directly from the installation folder. If you have multiple versions of e^2 studio installed, please make sure to launch the version of e^2 studio that was specified on the first page.
1.2	In the Eclipse Launcher window, specify the destination for the new workspace. It is recommended to keep the path simple and avoid using spaces.  Eclipse Launcher **Select a directory as workspace** e^2 studio uses the workspace directory to store its preferences and development artifacts. Workspace: C:\Workspaces\Renesas_RA Browse... □ Use this as the default and do not ask again **Recent Workspaces** Launch Cancel
1.3	Click **Launch** to start e^2 studio in the specified path. If prompted, press **Apply** to dismiss pop up window asking for permission to log and report usage (it will remain disabled).
1.4	The welcome screen will show inside the new workspace. It can be dismissed by clicking on the Hide button in the top-right corner. 
1.5	Go to **File -> Import** and Select **General -> CMSIS Pack**.
1.6	In the Import CMSIS Pack window, click **…** to browse for the .pack file containing BSP for VOICE-RA4E1 kit (Renesas.RA_board_ra4e1_voice.<version>.pack). Select **Renesas RA** from the drop-down box under Specify device family and click **Finish**.  Import CMSIS Pack **Import CMSIS Pack** Choose CMSIS pack to import Specify pack file: D:\Renesas.RA_board_ra4e1_voice.3.7.0.pack ... Specify device family: Renesas RA

1.7	Click **OK** in the pop-up window confirming successful pack file import.
1.8	Go to **File -> New** and select **Renesas C/C++ Project**, then **Renesas RA**.
1.9	In the new project wizard window, select **Renesas RA C/C++ Project** and click **Next**.  New C/C++ Project **Templates for Renesas RA Project** All C/C++ **Renesas RA C/C++ Project** *Create an executable or static library C/C++ project for Renesas RA.*
1.10	Specify a project name and Click **Next**.
1.11	Select FSP version matching your BSP and FSP installation (e.g., **3.7.0**) and set Board to **VOICE-RA4E1**. Verify that the Debugger is set to **J-Link ARM** and click **Next**.  Renesas RA C/C++ Project **Renesas RA C/C++ Project** Device and Tools Selection Device Selection FSP Version: 3.7.0 Board: VOICE-RA4E1 Device: R7FA4E10D2CNE Core: CM33 Language: ● C ○ C++ Board Description Voice User Interface for RA4E1 MCU Group Device Details TrustZone Yes Pins 48 Processor Cortex-M33 Toolchains GNU ARM Embedded 10.3.1.20210824 Debugger J-Link ARM < Back Next > Finish Cancel
1.12	Select **Flat (Non-TrustZone)** Project and click **Next**.
1.13	On the next window, leave **Executable** and **No RTOS** selected. Click **Next**.
1.14	On the final page of the new project wizard select **Bare Metal – Minimal** and click **Finish**.  Renesas RA C/C++ Project **Renesas RA C/C++ Project** Project Template Selection Project Template Selection ● **Bare Metal - Minimal** Bare metal FSP project that includes BSP. This project will initialize clocks, pins, stacks, and the C runtime environment. [Renesas.RA.3.7.0.pack]
1.15	When prompted to open the **FSP Configuration perspective**, click **Open Perspective**. The project is now set up to begin evaluation and development using the VOICE kit.

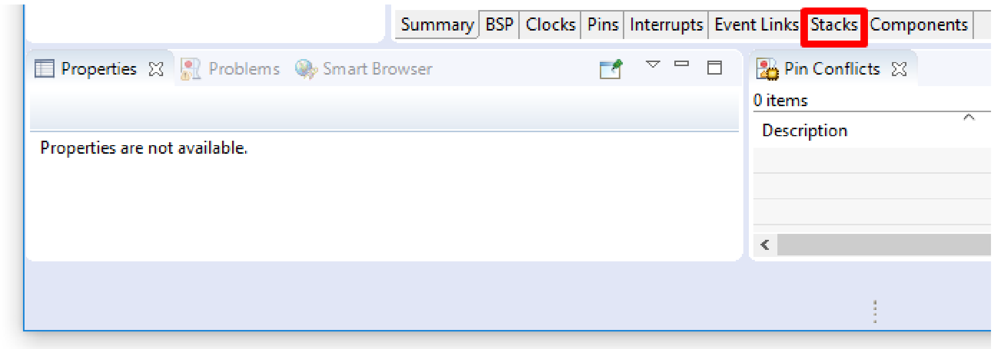
2 Configuring and using serial communications

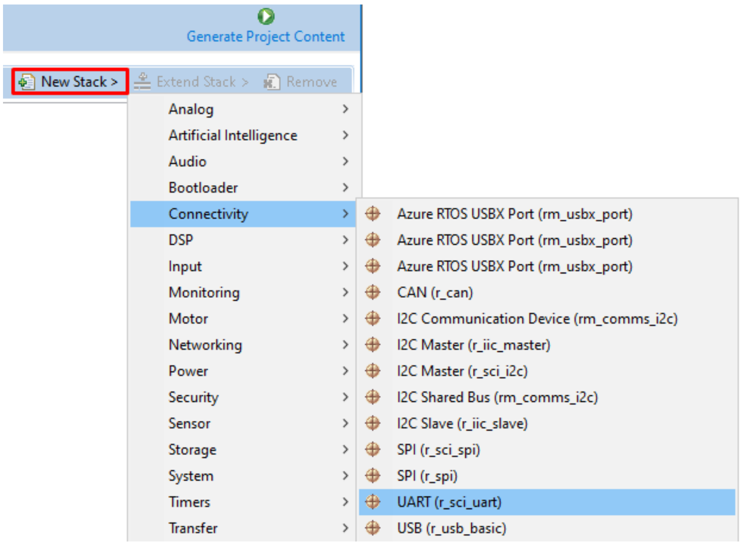
Overview

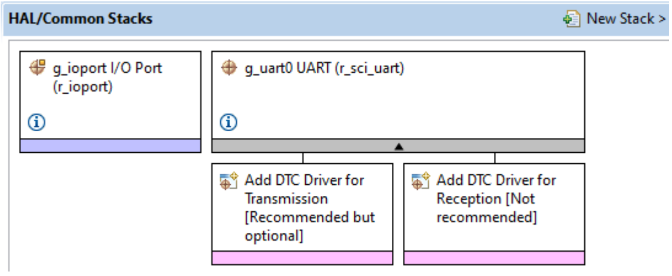
Following section explains how to configure and operate basic UART write and read functionality on the VOICE kit.

Procedural Steps

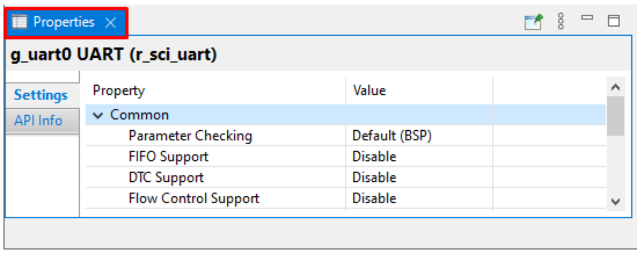
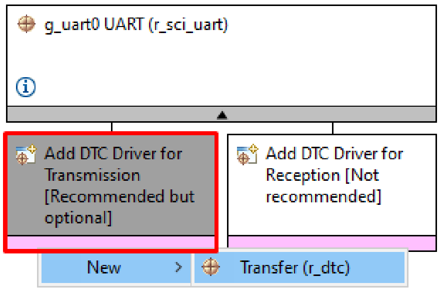
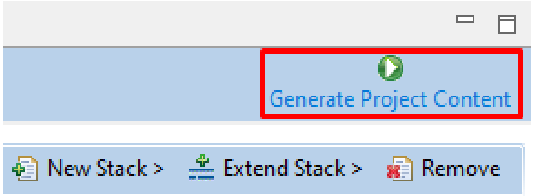
<table>
<tr><td>2.1</td><td>This section begins with an empty RA4E1-VOICE project. Make sure steps 1.8-1.15 are completed before starting this section.</td></tr>
<tr><td>2.2</td><td>Once new project is created, e^2 studio will switch to a layout optimized for developing Renesas RA projects. Select the <b>Stacks</b> tab at the bottom of the <b>FSP Configuration</b> pane visible in the middle.


</td></tr>
<tr><td>2.3</td><td>Open the <b>New Stack</b> menu (near the top-left corner) and navigate to <b>Connectivity -> UART (r_sci_uart)</b>.

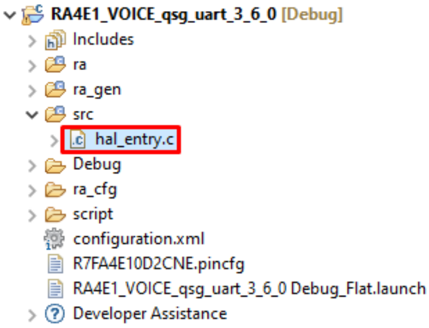

</td></tr>
<tr><td>2.4</td><td>A new driver module will be added inside the <b>HAL/Common group</b>.


HAL/Common Stacks
New Stack >
g_ioport I/O Port (r_ioport)
g_uart0 UART (r_sci_uart)
Add DTC Driver for Transmission [Recommended but optional]
Add DTC Driver for Reception [Not recommended]

</td></tr>
</table>

2.5	Click on **g_uart0 UART (r_sci_uart)** and go to the **Properties** tab. It can be found in the lower-left pane, directly under the **Project Explorer**.  Properties g_uart0 UART (r_sci_uart) Settings API Info Property — Value Common Parameter Checking — Default (BSP) FIFO Support — Disable DTC Support — Disable Flow Control Support — Disable
2.6	Set the following properties for g_uart0. You may need to expand the chevrons to access all of the properties: - Common -> FIFO Support — Enable - Common -> DTC Support — Enable - General -> Channel — 3 - Extra -> Receive FIFO Trigger Level — One - Interrupts -> Callback — g_uart0_cb
2.7	Click on Add DTC Driver for Transmission box underneath g_uart0 UART box and select New -> Transfer (r_dtc). All properties should be left unchanged for this module.  g_uart0 UART (r_sci_uart) Add DTC Driver for Transmission [Recommended but optional] Add DTC Driver for Reception [Not recommended] New Transfer (r_dtc)
2.8	RA Configuration for this section is complete. Apply changes to the project source by clicking the **Generate Project Content** button in the top-right corner of the Configurator window. When prompted to *Proceed with save and generate*, tick the box next to **Always save and generate without asking** and click **Proceed**.  Generate Project Content New Stack > Extend Stack > Remove
2.9	The FSP Configurator will extract all the necessary drivers and generate the code based on the configuration provided in the **Properties** tab.

2.10 In the **Project Explorer** pane, expand the **src** folder in the project and open **hal_entry.c**.



```
RA4E1_VOICE_qsg_uart_3_6_0 [Debug]
    Includes
    ra
    ra_gen
    src
        hal_entry.c
    Debug
    ra_cfg
    script
    configuration.xml
    R7FA4E10D2CNE.pincfg
    RA4E1_VOICE_qsg_uart_3_6_0 Debug_Flat.launch
    Developer Assistance
```

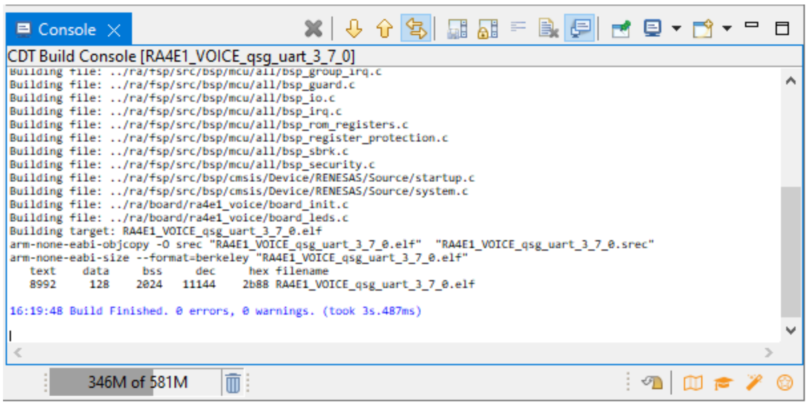
2.11 **hal_entry.c** contains user application entry point (hal_entry function) for RTOS-less projects. The `R_BSP_WarmStart` callback is provided for the user to specify additional functions to be called during the FSP initialization sequence (e.g., pin configuration).

2.12 **hal_entry.c** can be used to exercise API of the various modules configured inside FSP Configurator using Developer Assist or by writing code manually. Following code can be used to completely replace contents of hal_entry.c to perform basic UART write and read operations:

```
#include "hal_data.h"
#include "stdio.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER

static volatile bool uart_done;
static volatile char uart_rec;

void hal_entry(void)
{
    fsp_err_t err;

    /* Initialize SCI peripheral in UART mode */
    err = R_SCI_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Perform UART write */
    err = R_SCI_UART_Write(&g_uart0_ctrl, (void *) "Hello from Renesas VOICE kit\r\n", 30);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Wait for interrupt & check for completion */
    while (false == uart_done)
        __WFI();

    uart_done = false;

    while (1)
    {
        /* Wait for interrupt & check for received data */
        while ('\0' == uart_rec)
            __WFI();

        char text_buf[32] = {0};
        snprintf(text_buf, 32, "Received character: '%c'\r\n", uart_rec);

        uart_rec = '\0';

        /* Perform UART write */
        err = R_SCI_UART_Write(&g_uart0_ctrl, (void *) text_buf, strlen(text_buf));
        if (FSP_SUCCESS != err)
        {
            __BKPT(0);
        }

        /* Wait for interrupt & check for completion */
        while (false == uart_done)
            __WFI();

        uart_done = false;
    }
}

void g_uart0_cb(uart_callback_args_t * p_args)
{
    if (UART_EVENT_TX_COMPLETE == p_args->event)
    {
        uart_done = true;
    }

    else if (UART_EVENT_RX_CHAR == p_args->event)
    {
        uart_rec = (char) p_args->data;
    }

    else
    {}
}

void R_BSP_WarmStart(bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_POST_C == event)
    {
        /* C runtime environment and system clocks are setup. */

        /* Configure pins. */
        R_IOPORT_Open (&g_ioport_ctrl, g_ioport.p_cfg);
    }
}
```

2.13 The project is now ready to compile. Press the “hammer” icon to start building the project.

2.14 Once the build has finished, the console pane in the lower-right corner of e^2 studio will report zero error and warnings:

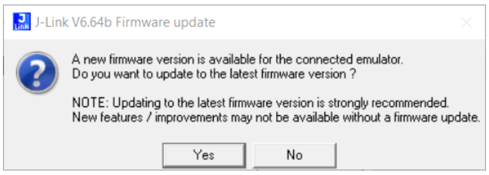


CDT Build Console [RA4E1_VOICE_qsg_uart_3_7_0]

```
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_group_irq.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_guard.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_io.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_irq.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_rom_registers.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_register_protection.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_sbrk.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_security.c
Building file: ../ra/fsp/src/bsp/cmsis/Device/RENESAS/Source/startup.c
Building file: ../ra/fsp/src/bsp/cmsis/Device/RENESAS/Source/system.c
Building file: ../ra/board/ra4e1_voice/board_init.c
Building file: ../ra/board/ra4e1_voice/board_leds.c
Building target: RA4E1_VOICE_qsg_uart_3_7_0.elf
arm-none-eabi-objcopy -O srec "RA4E1_VOICE_qsg_uart_3_7_0.elf"  "RA4E1_VOICE_qsg_uart_3_7_0.srec"
arm-none-eabi-size --format=berkeley "RA4E1_VOICE_qsg_uart_3_7_0.elf"
   text    data     bss     dec     hex filename
   8992     128    2024   11144    2b88 RA4E1_VOICE_qsg_uart_3_7_0.elf

16:19:48 Build Finished. 0 errors, 0 warnings. (took 3s.487ms)
```

2.15 The application is now ready to be programmed and run on the VOICE kit. Press the “bug” icon to begin the debug session.

2.16 You may be prompted to update the J-Link debugger firmware. You can click **Yes** to update. It will take a few moments to complete.

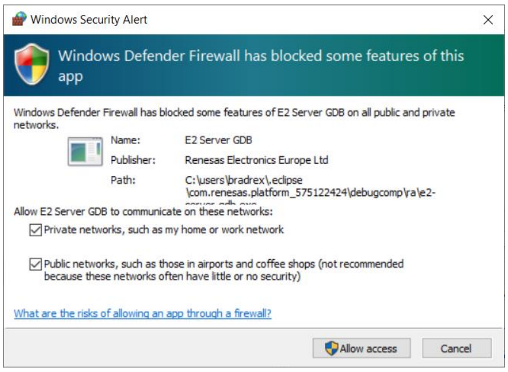


J-Link V6.64b Firmware update

A new firmware version is available for the connected emulator.
Do you want to update to the latest firmware version ?

NOTE: Updating to the latest firmware version is strongly recommended.
New features / improvements may not be available without a firmware update.

2.17 Windows could also prompt you to allow the GDB server through your firewall. Click the checkbox to allow it through private networks, then **Allow** access.



Windows Security Alert

Windows Defender Firewall has blocked some features of this app

Windows Defender Firewall has blocked some features of E2 Server GDB on all public and private networks.

Name: E2 Server GDB
Publisher: Renesas Electronics Europe Ltd
Path: C:\users\bradrex\.eclipse
\com.renesas.platform_575122424\debugcomp\ra\e2-

Allow E2 Server GDB to communicate on these networks:

☑ Private networks, such as my home or work network

☑ Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)

What are the risks of allowing an app through a firewall?

2.18 e^2 studio will perform flash programming routines and prompt to switch to **Debug** perspective. Select the check box by **Remember my decision** and click **Switch**.

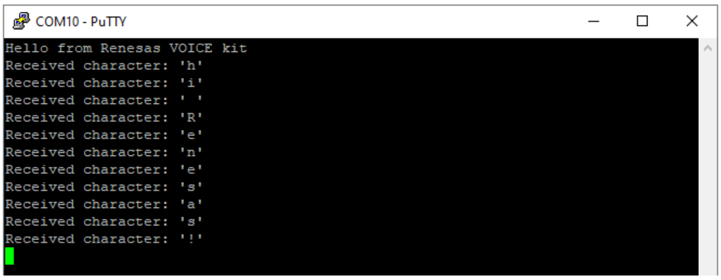
2.19 The debug session is now started, and the application is paused at its entry function (`SystemInit()` in `Reset_Handler`). At this point, you can set up additional debug features such as variable and expressions views before the program is executed.

2.20 Renesas VOICE kits include an on-board debugger with USB-to-UART functionality. Open the serial terminal program of your choice (e.g. PuTTY or TeraTerm) to communicate with the UART interface configured earlier in this section (use the device manager to identify the correct COM port if needed, set baud rate to 115200). The Virtual COM (VCOM) port will stay live as long as the kit is connected to the host, even when the debug session has been terminated or the MCU has been reset.

2.21 Click the **Resume** button or press **F8** on the keyboard to start the application.

2.22 The Program will stop again, this time at the start of the main function. Low-level initialization routines are now completed. Press **Resume** or **F8** again to resume the application and begin executing user code.

2.23 Go back to the serial terminal to observe the output from the VOICE kit. Experiment with various keyboard inputs to exercise UART read and write functionality (screenshot below shows PuTTY):



COM10 - PuTTY

```
Hello from Renesas VOICE kit
Received character: 'h'
Received character: 'i'
Received character: ' '
Received character: 'R'
Received character: 'e'
Received character: 'n'
Received character: 'e'
Received character: 's'
Received character: 'a'
Received character: 's'
Received character: '!'
```

2.24	Click the **Terminate** button or press **Ctrl + F2** on the keyboard to stop the application and terminate the debug session. 🟥

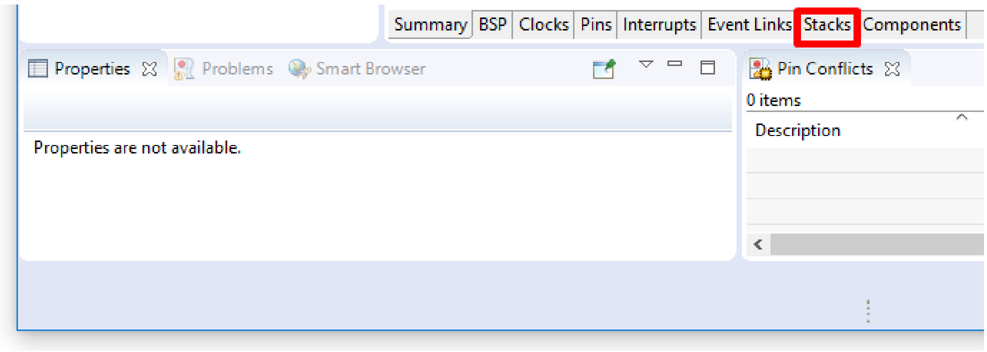
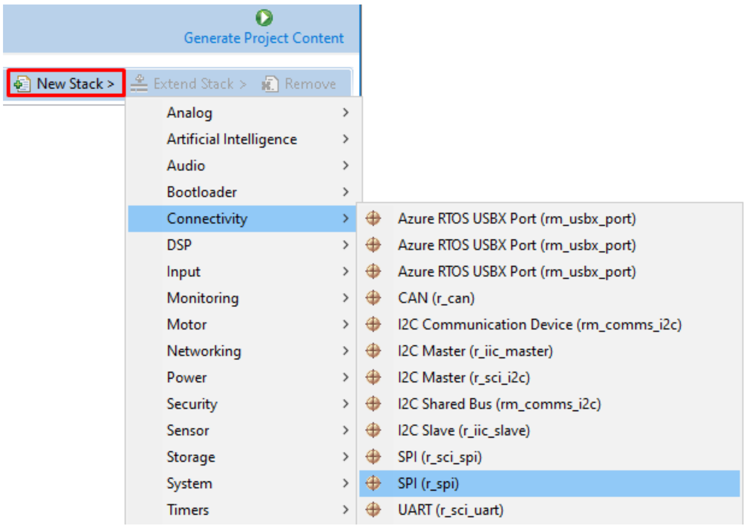
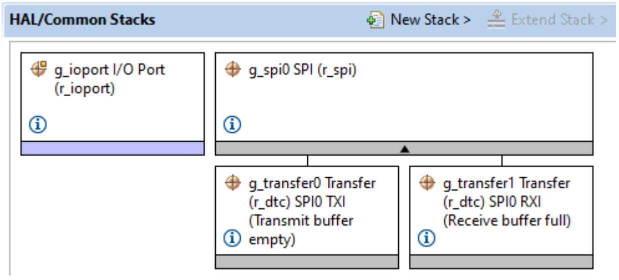
END OF SECTION

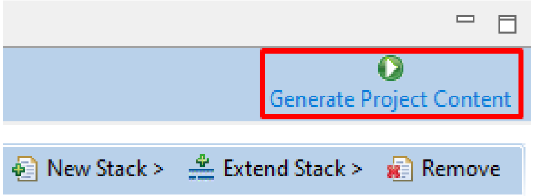
3 Configuring and using digital microphones

Overview

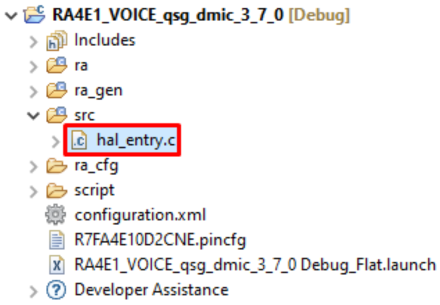
Following section explains how to configure and operate a digital microphone to capture audio input on the VOICE kit.

Procedural Steps

3.1	This section begins with an empty RA4E1-VOICE project. Make sure steps 1.8-1.15 are completed before starting this section.
3.2	Once new project is created, e^2 studio will switch to a layout optimized for developing Renesas RA projects. Select the **Stacks** tab at the bottom of the **FSP Configuration** pane visible in the middle.  Summary BSP Clocks Pins Interrupts Event Links Stacks Components Properties Problems Smart Browser Pin Conflicts 0 items Description Properties are not available.
3.3	Open the **New Stack** menu (near the top-left corner) and navigate to **Connectivity -> SPI (r_spi)**.  Generate Project Content New Stack > Extend Stack > Remove Analog Artificial Intelligence Audio Bootloader Connectivity DSP Input Monitoring Motor Networking Power Security Sensor Storage System Timers Azure RTOS USBX Port (rm_usbx_port) Azure RTOS USBX Port (rm_usbx_port) Azure RTOS USBX Port (rm_usbx_port) CAN (r_can) I2C Communication Device (rm_comms_i2c) I2C Master (r_iic_master) I2C Master (r_sci_i2c) I2C Shared Bus (rm_comms_i2c) I2C Slave (r_iic_slave) SPI (r_sci_spi) SPI (r_spi) UART (r_sci_uart)
3.4	A new driver module will be added inside the **HAL/Common group**. Unlike some other drivers (e.g. UART), DTC modules will be populated automatically.  HAL/Common Stacks New Stack > Extend Stack > g_ioport I/O Port (r_ioport) g_spi0 SPI (r_spi) g_transfer0 Transfer (r_dtc) SPI0 TXI (Transmit buffer empty) g_transfer1 Transfer (r_dtc) SPI0 RXI (Receive buffer full)

3.5	Click on **g_spi0 SPI (r_spi)**, go to the **Properties** tab and apply the following settings. You may need to expand the chevrons to access all of the properties: • Operating Mode Slave • Callback g_spi0_cb • SPI Mode SPI Operation
3.6	Access the **New Stack** menu again and select **Timers -> Timer, General PWM (r_gpt)**. Use **Properties** tab to configure following properties for this new module: • Common -> Pin Output Support Enabled • General -> Name g_timer_bclk • General -> Channel 1 • General -> Period 1024000 • General -> Period Unit Hertz • Output -> GTIOCB Output Enabled True
3.7	Access the **New Stack** menu yet again and select **Timers -> Timer, General PWM (r_gpt)**. Use **Properties** tab to configure following properties for this new module: • General -> Name g_timer_ws • General -> Channel 4 • General -> Mode PWM • General -> Period 32 • General -> Period Unit Raw Counts • Output -> Custom Waveform -> GTIOB -> Initial Output Level Pin Level High • Output -> Custom Waveform -> GTIOB -> Compare Match Output Level Pin Level Toggle • Output -> Custom Waveform -> Custom Waveform Enable Enabled • Output -> GTIOCB Output Enabled True • Input -> Count Up Source GPT1 COUNTER OVERFLOW (check the box).
3.8	g_timer_ws is highlighted in red to indicate that configuring g_timer_ws to count overflows of g_timer_bclk requires ELC driver. Use **New Stack** menu and navigate to **System -> Event Link Controller (r_elc)**. No configuration is needed for this module.
3.9	RA Configuration for this section is complete. Apply changes to the project source by clicking the **Generate Project Content** button in the top-right corner of the Configurator window. When prompted to *Proceed with save and generate*, tick the box next to **Always save and generate without asking** and click **Proceed**.  Generate Project Content New Stack > Extend Stack > Remove
3.10	The FSP Configurator will extract all the necessary drivers and generate the code based on the configuration provided in the **Properties** tab.

3.11 In the **Project Explorer** pane, expand the **src** folder in the project and open **hal_entry.c**.



```
RA4E1_VOICE_qsg_dmic_3_7_0 [Debug]
  Includes
  ra
  ra_gen
  src
    hal_entry.c
  ra_cfg
  script
  configuration.xml
  R7FA4E10D2CNE.pincfg
  RA4E1_VOICE_qsg_dmic_3_7_0 Debug_Flat.launch
  Developer Assistance
```

3.12 **hal_entry.c** contains user application entry point (hal_entry function) for RTOS-less projects. The `R_BSP_WarmStart` callback is provided for the user to specify additional functions to be called during the FSP initialization sequence (e.g., pin configuration).

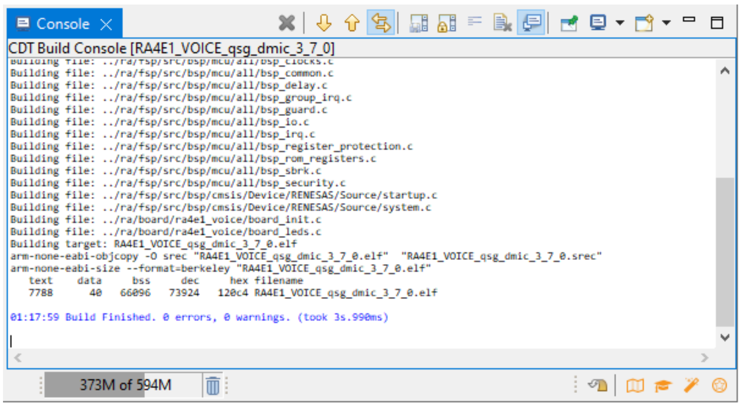
3.13 **hal_entry.c** can be used to exercise API of the various modules configured inside FSP Configurator using Developer Assist or by writing code manually. Following code can be used to completely replace contents of hal_entry.c to perform sound capture using the digital microphone on the VOICE kit:

```
#include "hal_data.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER

#define DMIC_BUF_SIZE   (8000)

static uint32_t dmic_buf[2][DMIC_BUF_SIZE];
static volatile uint8_t dmic_idx;

static volatile bool dmic_done;
static volatile bool dmic_err;

void hal_entry(void)
{
    fsp_err_t err;

    /* Initialize ELC peripheral */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Enabled configured ELC links */
    err = R_ELC_Enable(&g_elc_ctrl);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Initialize timer used to generate I2S BLCK signal */
    err = R_GPT_Open(&g_timer_bclk_ctrl, &g_timer_bclk_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Initialize timer used to generate I2S WS signal */
    err = R_GPT_Open(&g_timer_ws_ctrl, &g_timer_ws_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Set initial counter value before the cycle start for SPI to register falling edge */
    err = R_GPT_CounterSet(&g_timer_ws_ctrl, g_timer_ws_cfg.period_counts - 1);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Enable the I2S WS timer (counting will only start after I2S BLCK is enabled) */
    err = R_GPT_Start(&g_timer_ws_ctrl);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Initialize SPI perpiheral used to receive I2S data */
    err = R_SPI_Open(&g_spi0_ctrl, &g_spi0_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Start the I2S BCLK clock */
    err = R_GPT_Start(&g_timer_bclk_ctrl);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Set up the initial I2S read */
    err = R_SPI_Read(&g_spi0_ctrl, dmic_buf[dmic_idx], DMIC_BUF_SIZE, SPI_BIT_WIDTH_32_BITS);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    while (1)
    {
        /* Wait for interrupt & check for event */
        while ((false == dmic_done) && (false == dmic_err))
            __WFI();

        if (true == dmic_err)
        {
            dmic_err = false;

            /* Restart SPI peripheral to clear the underrun error state */
            R_SPI_Close(&g_spi0_ctrl);
            R_SPI_Open(&g_spi0_ctrl, &g_spi0_cfg);
            do
            {
                /* Repeat this request if it fails */
                err = R_SPI_Read(&g_spi0_ctrl, dmic_buf, DMIC_BUF_SIZE, SPI_BIT_WIDTH_32_BITS);
            }
            while (FSP_SUCCESS != err);
        }

        else // (true == dmic_done)
        {
            dmic_done = false;

            /* Trim and align data down to 16-bit */
            for (int i = 0; i < DMIC_BUF_SIZE; i++)
            {
                dmic_buf[dmic_idx ^ 1][i] = (dmic_buf[dmic_idx ^ 1][i] >> 15) & 0xFFFF;
            }

            /** Data in dmic_buf[dmic_idx ^ 1] can be used at this point */

            /* Toggle blue LED to indicate buffer received */
            bsp_io_level_t level;
            R_IOPORT_PinRead(&g_ioport_ctrl, BSP_IO_PORT_02_PIN_12, &level);
            R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_02_PIN_12, !level);
        }
    }
}

void g_spi0_cb(spi_callback_args_t * p_args)
```

```
{
    if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
        /* Change index of the active write buffer */
        dmic_idx ^= 1;

        /* Start subsequent I2S read */
        R_SPI_Read(&g_spi0_ctrl, dmic_buf[dmic_idx], DMIC_BUF_SIZE, SPI_BIT_WIDTH_32_BITS);

        dmic_done = true;
    }

    else if (SPI_EVENT_ERR_MODE_UNDERRUN == p_args->event)
    {
        /* SPI peripheral wasn't ready when data was sent */
        dmic_err = true;
    }

    else
    {}
}

void R_BSP_WarmStart(bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_POST_C == event)
    {
        /* C runtime environment and system clocks are setup. */

        /* Configure pins. */
        R_IOPORT_Open (&g_ioport_ctrl, g_ioport.p_cfg);
    }
}
```

3.14 The project is now ready to compile. Press the “hammer” icon to start building the project.

3.15 Once the build has finished, the console pane in the lower-right corner of e^2 studio will report zero error and warnings:



CDT Build Console [RA4E1_VOICE_qsg_dmic_3_7_0]

```
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_clocks.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_common.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_delay.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_group_irq.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_guard.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_io.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_irq.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_register_protection.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_rom_registers.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_sbrk.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_security.c
Building file: ../ra/fsp/src/bsp/cmsis/Device/RENESAS/Source/startup.c
Building file: ../ra/fsp/src/bsp/cmsis/Device/RENESAS/Source/system.c
Building file: ../ra/board/ra4e1_voice/board_init.c
Building file: ../ra/board/ra4e1_voice/board_leds.c
Building target: RA4E1_VOICE_qsg_dmic_3_7_0.elf
arm-none-eabi-objcopy -O srec "RA4E1_VOICE_qsg_dmic_3_7_0.elf"  "RA4E1_VOICE_qsg_dmic_3_7_0.srec"
arm-none-eabi-size --format=berkeley "RA4E1_VOICE_qsg_dmic_3_7_0.elf"
   text    data     bss     dec     hex filename
   7788      40   66096   73924   120c4 RA4E1_VOICE_qsg_dmic_3_7_0.elf

01:17:59 Build Finished. 0 errors, 0 warnings. (took 3s.990ms)
```

3.16 The application is now ready to be programmed and run on the VOICE kit. Press the “bug” icon to begin the debug session.

3.17 You may be prompted to update the J-Link debugger firmware. You can click **Yes** to update. It will take a few moments to complete.

3.18 Windows could also prompt you to allow the GDB server through your firewall. Click the checkbox to allow it through private networks, then **Allow** access.

3.19	e^2 studio will perform flash programming routines and prompt to switch to **Debug** perspective. Select the check box by **Remember my decision** and click **Switch**.
3.20	The debug session is now started, and the application is paused at its entry function (`SystemInit()` in `Reset_Handler`). At this point, you can set up additional debug features such as variable and expressions views before the program is executed.
3.21	Click the **Resume** button or press **F8** on the keyboard to start the application. ▶
3.22	The Program will stop again, this time at the start of the main function. Low-level initialization routines are now completed. Press **Resume** or **F8** again to resume the application and begin executing user code.
3.23	As application is executing, the blue LED will toggle each time a new audio buffer is captured. The sound capture is running continuously with each new data set being passed to the main loop approximately every 500ms (16000Hz sampling rate with 8000 samples per buffer). The sample code implements double buffering to allow for further processing of the data without breaking the data continuity. Example application can be easily extended to use the data captured, e.g. for voice recognition model or real-time streaming to another host.
3.24	Click the **Terminate** button or press **Ctrl + F2** on the keyboard to stop the application and terminate the debug session. ■

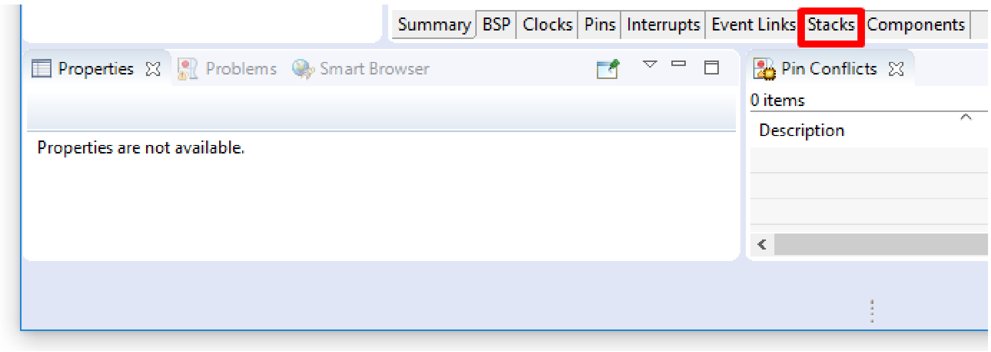
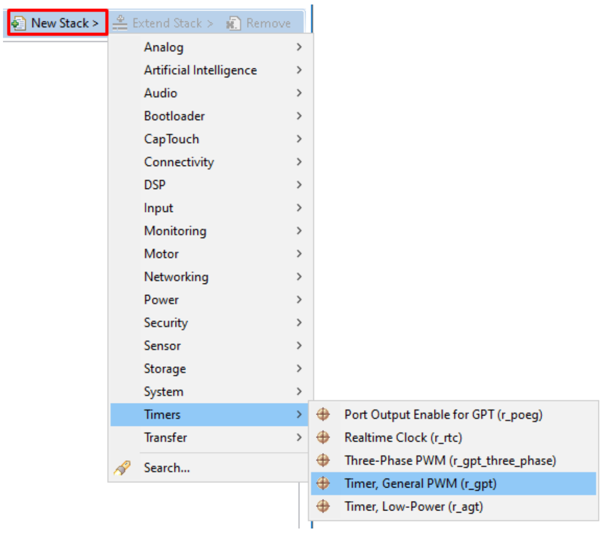
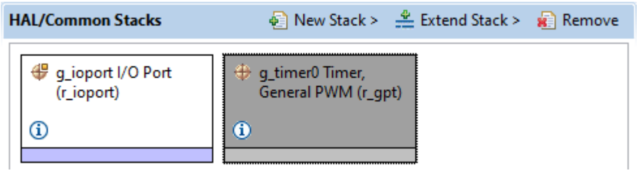
END OF SECTION

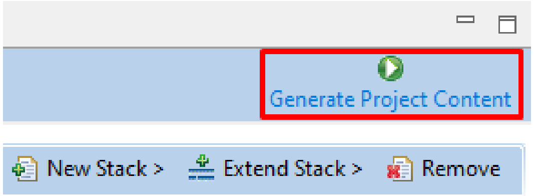
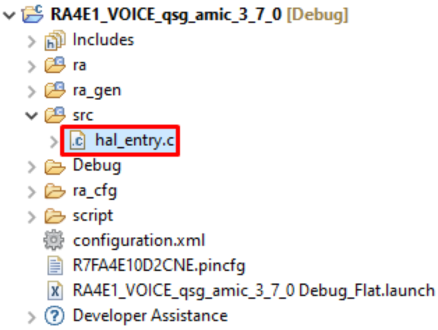
4 Configuring and using analog microphones

Overview

Following section explains how to configure and operate a pair of analog microphones to capture audio input on the VOICE kit.

Procedural Steps

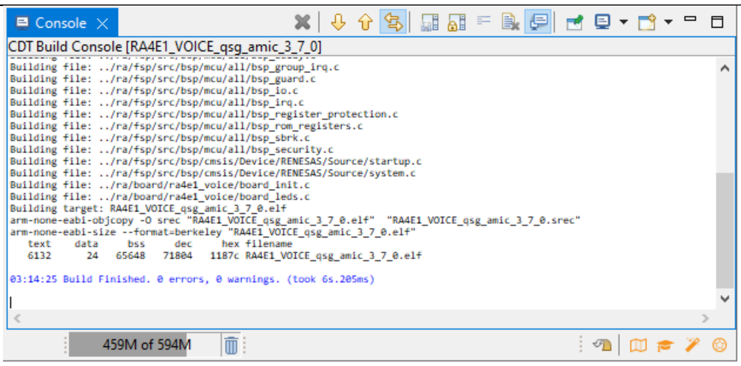
4.1	This section begins with an empty RA4E1-VOICE project. Make sure steps 1.8-1.15 are completed before starting this section.
4.2	Once new project is created, e^2 studio will switch to a layout optimized for developing Renesas RA projects. Select the **Stacks** tab at the bottom of the **FSP Configuration** pane visible in the middle.  Summary BSP Clocks Pins Interrupts Event Links Stacks Components Properties Problems Smart Browser Pin Conflicts 0 items Description Properties are not available.
4.3	Open the **New Stack** menu and navigate to **Timers -> Timer, General PWM (r_gpt)**.  New Stack > Extend Stack > Remove Analog Artificial Intelligence Audio Bootloader CapTouch Connectivity DSP Input Monitoring Motor Networking Power Security Sensor Storage System Timers Transfer Search... Port Output Enable for GPT (r_poeg) Realtime Clock (r_rtc) Three-Phase PWM (r_gpt_three_phase) Timer, General PWM (r_gpt) Timer, Low-Power (r_agt)
4.4	A new driver module will be added inside the **HAL/Common group**.  HAL/Common Stacks New Stack > Extend Stack > Remove g_ioport I/O Port (r_ioport) g_timer0 Timer, General PWM (r_gpt)

4.5	Click on **g_timer0 Timer, General PWM (r_gpt)**, go to the **Properties** tab and apply the following settings. You may need to expand the chevrons to access all of the properties: • General -> Channel 2 • General -> Period 16000 • General -> Period Unit Hertz
4.6	Access the **New Stack** menu again and select **Analog -> ADC (r_adc)**. Use **Properties** tab to configure following properties for this new module: • Input -> Channel Scan Mask Channel 0 + Channel 1 (check both boxes) • Interrupts -> Normal/Group A Trigger GPT**2** COUNTER OVERFLOW
4.7	Access the **New Stack** menu yet again and select **Transfer -> Transfer (r_dmac)**. Use **Properties** tab to configure following properties for this new module: • Transfer Size 4 Bytes • Destination Address Mode Incremented • Activation Source ADC0 SCAN END • Callback g_transfer0_cb • Transfer End Interrupt Priority Priority 11
4.8	g_adc0 is highlighted in red to indicate that configuring g_adc0 to trigger ADC conversion on timer overflow requires ELC driver. Use **New Stack** menu and navigate to **System -> Event Link Controller (r_elc)**. No configuration is needed for this module.
4.9	RA Configuration for this section is complete. Apply changes to the project source by clicking the **Generate Project Content** button in the top-right corner of the Configurator window. When prompted to *Proceed with save and generate*, tick the box next to **Always save and generate without asking** and click **Proceed**.  Generate Project Content New Stack > Extend Stack > Remove
4.10	The FSP Configurator will extract all the necessary drivers and generate the code based on the configuration provided in the **Properties** tab.
4.11	In the **Project Explorer** pane, expand the **src** folder in the project and open **hal_entry.c**.  RA4E1_VOICE_qsg_amic_3_7_0 [Debug] Includes ra ra_gen src hal_entry.c Debug ra_cfg script configuration.xml R7FA4E10D2CNE.pincfg RA4E1_VOICE_qsg_amic_3_7_0 Debug_Flat.launch Developer Assistance
4.12	**hal_entry.c** contains user application entry point (hal_entry function) for RTOS-less projects. The `R_BSP_WarmStart` callback is provided for the user to specify additional functions to be called during the FSP initialization sequence (e.g., pin configuration).

4.13 **hal_entry.c** can be used to exercise API of the various modules configured inside FSP Configurator using Developer Assist or by writing code manually. Following code can be used to completely replace contents of hal_entry.c to perform sound capture using the digital microphone on the VOICE kit:

```
#include "hal_data.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER

#define AMIC_BUF_SIZE   (8000)

static uint32_t amic_buf[2][AMIC_BUF_SIZE];
static volatile uint8_t amic_idx;

static volatile bool amic_done;

void hal_entry(void)
{
    fsp_err_t err;

    /* Initialize ELC peripheral */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Enabled configured ELC links */
    err = R_ELC_Enable(&g_elc_ctrl);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Initialize the ADC peripheral */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Enable ADC scanning on microphone channels */
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Enable ADC scanning */
    err = R_ADC_ScanStart(&g_adc0_ctrl);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Initialize the DMA peripheral */
    err = R_DMAC_Open(&g_transfer0_ctrl, &g_transfer0_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Set the DMA to capture from ADC registers into amic_buf */
    err = R_DMAC_Reset(&g_transfer0_ctrl, (void *) R_ADC0->ADDR, amic_buf[amic_idx],
AMIC_BUF_SIZE);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Initialize timer used to limit the sampling rate */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Start the timer */
    err = R_GPT_Start(&g_timer0_ctrl);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    while (1)
    {
        /* Wait for interrupt & check for event */
        while (false == amic_done)
            __WFI();

        amic_done = false;

        /** Data in amic_buf[amic_idx ^ 1] can be used at this point */

        /* Toggle green LED to indicate buffer received */
        bsp_io_level_t level;
        R_IOPORT_PinRead(&g_ioport_ctrl, BSP_IO_PORT_02_PIN_13, &level);
        R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_02_PIN_13, !level);
    }
}

void g_transfer0_cb(dmac_callback_args_t * p_args)
{
    /* Change index of the active write buffer */
    amic_idx ^= 1;

    /* Start subsequent ADC capture */
    R_DMAC_Reset(&g_transfer0_ctrl, (void *) R_ADC0->ADDR, amic_buf[amic_idx], AMIC_BUF_SIZE);

    amic_done = true;

    /* Suppress compiler warning for unused p_args */
    FSP_PARAMETER_NOT_USED(p_args);
}

void R_BSP_WarmStart(bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_POST_C == event)
    {
        /* C runtime environment and system clocks are setup. */

        /* Configure pins. */
        R_IOPORT_Open (&g_ioport_ctrl, g_ioport.p_cfg);
    }
}
```

4.14	The project is now ready to compile. Press the “hammer” icon to start building the project. 🔨 ▾
4.15	Once the build has finished, the console pane in the lower-right corner of e^2 studio will report zero error and warnings:



CDT Build Console [RA4E1_VOICE_qsg_amic_3_7_0]

```
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_group_irq.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_guard.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_io.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_irq.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_register_protection.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_rom_registers.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_sbrk.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_security.c
Building file: ../ra/fsp/src/bsp/cmsis/Device/RENESAS/Source/startup.c
Building file: ../ra/fsp/src/bsp/cmsis/Device/RENESAS/Source/system.c
Building file: ../ra/board/ra4e1_voice/board_init.c
Building file: ../ra/board/ra4e1_voice/board_leds.c
Building target: RA4E1_VOICE_qsg_amic_3_7_0.elf
arm-none-eabi-objcopy -O srec "RA4E1_VOICE_qsg_amic_3_7_0.elf"  "RA4E1_VOICE_qsg_amic_3_7_0.srec"
arm-none-eabi-size --format=berkeley "RA4E1_VOICE_qsg_amic_3_7_0.elf"
   text    data     bss     dec     hex filename
   6132      24   65648   71804   1187c RA4E1_VOICE_qsg_amic_3_7_0.elf

03:14:25 Build Finished. 0 errors, 0 warnings. (took 6s.205ms)
```

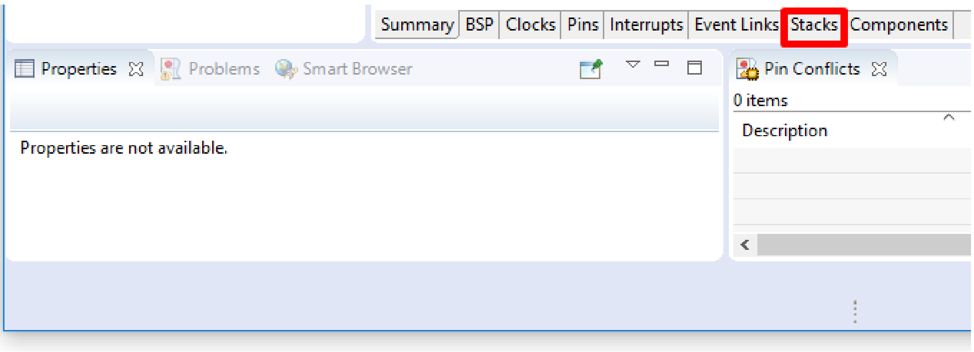
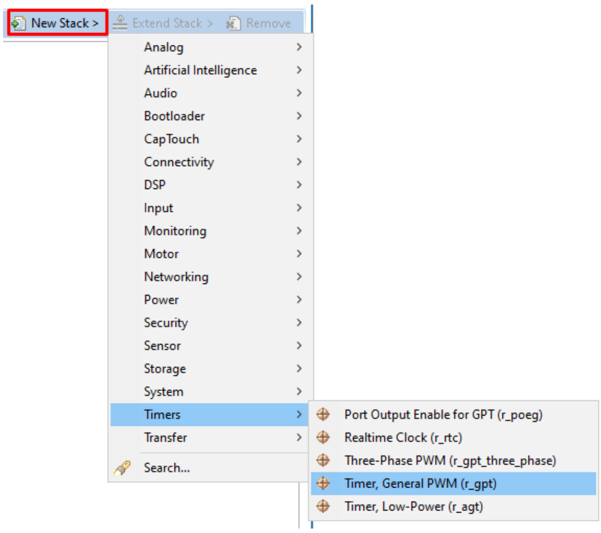
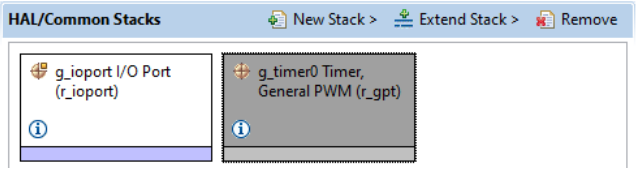
4.16	The application is now ready to be programmed and run on the VOICE kit. Press the "bug" icon to begin the debug session.
4.17	You may be prompted to update the J-Link debugger firmware. You can click **Yes** to update. It will take a few moments to complete.
4.18	Windows could also prompt you to allow the GDB server through your firewall. Click the checkbox to allow it through private networks, then **Allow** access.
4.19	e^2 studio will perform flash programming routines and prompt to switch to **Debug** perspective. Select the check box by **Remember my decision** and click **Switch**.
4.20	The debug session is now started, and the application is paused at its entry function (`SystemInit()` in `Reset_Handler`). At this point, you can set up additional debug features such as variable and expressions views before the program is executed.
4.21	Click the **Resume** button or press **F8** on the keyboard to start the application.
4.22	The Program will stop again, this time at the start of the main function. Low-level initialization routines are now completed. Press **Resume** or **F8** again to resume the application and begin executing user code.
4.23	As application is executing, the green LED will toggle each time a new audio buffer is captured. The sound capture is running continuously with each new data set being passed to the main loop approximately every 500ms (16000Hz sampling rate with 8000 samples per buffer). The sample code implements double buffering to allow for further processing of the data without breaking the data continuity. Example application can be easily extended to use the data captured, e.g. for voice recognition model or real-time streaming to another host.
4.24	Click the **Terminate** button or press **Ctrl + F2** on the keyboard to stop the application and terminate the debug session.

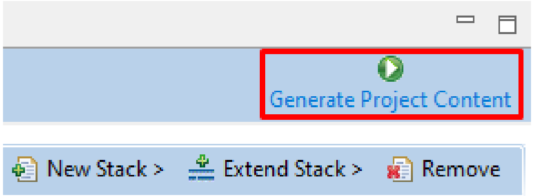
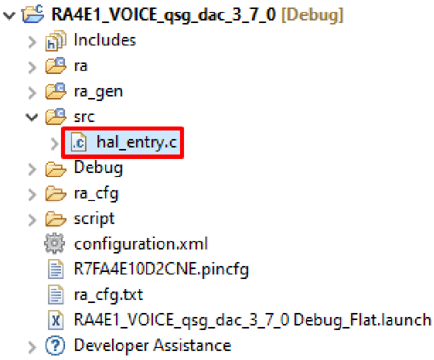
5 Configuring and using audio output

Overview

Following section explains how to configure and operate an on-chip DAC to output audio on the VOICE kit.

Procedural Steps

5.1	This section begins with an empty RA4E1-VOICE project. Make sure steps 1.8-1.15 are completed before starting this section.
5.2	Once new project is created, e^2 studio will switch to a layout optimized for developing Renesas RA projects. Select the **Stacks** tab at the bottom of the **FSP Configuration** pane visible in the middle.  Summary BSP Clocks Pins Interrupts Event Links Stacks Components Properties Problems Smart Browser Pin Conflicts 0 items Description Properties are not available.
5.3	Open the **New Stack** menu and navigate to **Timers -> Timer, General PWM (r_gpt)**.  New Stack > Extend Stack > Remove Analog Artificial Intelligence Audio Bootloader CapTouch Connectivity DSP Input Monitoring Motor Networking Power Security Sensor Storage System Timers Transfer Search... Port Output Enable for GPT (r_poeg) Realtime Clock (r_rtc) Three-Phase PWM (r_gpt_three_phase) Timer, General PWM (r_gpt) Timer, Low-Power (r_agt)
5.4	A new driver module will be added inside the **HAL/Common group**.  HAL/Common Stacks New Stack > Extend Stack > Remove g_ioport I/O Port (r_ioport) g_timer0 Timer, General PWM (r_gpt)

5.5	Click on **g_timer0 Timer, General PWM (r_gpt)**, go to the **Properties** tab and apply the following settings. You may need to expand the chevrons to access all of the properties: • General -> Name g_timer_dac* • General -> Channel 5* • General -> Period 16000 • General -> Period Unit Hertz * Changing timer instance name and channel number is recommended to avoid name conflict with other timer instances that might be used in the project (e.g. for analog and digital microphones). Timer period should match the sampling rate for the sounds to play back.
5.6	Access the **New Stack** menu again and select **Analog -> DAC (r_dac)**. Use **Properties** tab to configure following properties for this new module: • Data Format Left Justified
5.7	Access the **New Stack** menu yet again and select **Transfer -> Transfer (r_dmac)**. Use **Properties** tab to configure following properties for this new module: • Name g_transfer_dac* • Channel 1* • Source Address Mode Incremented • Activation Source GPT5 COUNTER OVERFLOW • Callback g_transfer_dac_cb • Transfer End Interrupt Priority Priority 13 * Changing DMAC instance name and channel number is recommended to avoid name conflict with other DMAC instances that might be used in the project (e.g. used for analog microphones).
5.8	RA Configuration for this section is complete. Apply changes to the project source by clicking the **Generate Project Content** button in the top-right corner of the Configurator window. When prompted to *Proceed with save and generate*, tick the box next to **Always save and generate without asking** and click **Proceed**.  Generate Project Content New Stack > Extend Stack > Remove
5.9	The FSP Configurator will extract all the necessary drivers and generate the code based on the configuration provided in the **Properties** tab.
5.10	In the **Project Explorer** pane, expand the **src** folder in the project and open **hal_entry.c**.  RA4E1_VOICE_qsg_dac_3_7_0 [Debug] Includes ra ra_gen src hal_entry.c Debug ra_cfg script configuration.xml R7FA4E10D2CNE.pincfg ra_cfg.txt RA4E1_VOICE_qsg_dac_3_7_0 Debug_Flat.launch Developer Assistance

5.11	**hal_entry.c** contains user application entry point (hal_entry function) for RTOS-less projects. The `R_BSP_WarmStart` callback is provided for the user to specify additional functions to be called during the FSP initialization sequence (e.g., pin configuration).
5.12	**hal_entry.c** can be used to exercise API of the various modules configured inside FSP Configurator using Developer Assist or by writing code manually. Following code can be used to completely replace contents of hal_entry.c to perform sound capture using the digital microphone on the VOICE kit:

```
#include "hal_data.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER

extern uint8_t audio_samples[130032];

static volatile bool dac_done;

void hal_entry(void)
{
    fsp_err_t err;

    /* Initialize the DAC peripheral */
    err = R_DAC_Open(&g_dac0_ctrl, &g_dac0_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Enable DAC output */
    err = R_DAC_Start(&g_dac0_ctrl);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Initialize the DMA peripheral */
    err = R_DMAC_Open(&g_transfer_dac_ctrl, &g_transfer_dac_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Initialize the timer used to control the sampling rate */
    err = R_GPT_Open(&g_timer_dac_ctrl, &g_timer_dac_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Start the timer */
    err = R_GPT_Start(&g_timer_dac_ctrl);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    while (1)
    {
        /* Start playback by setting DMA to transfer audio samples to the DAC */
        err = R_DMAC_Reset(&g_transfer_dac_ctrl, audio_samples, (void *) R_DAC->DADR,
                           sizeof(audio_samples) / sizeof(uint16_t));
        if (FSP_SUCCESS != err)
        {
            __BKPT(0);
        }

        /* Wait for interrupt & check for event */
        while (false == dac_done)
            __WFI();

        dac_done = false;
        /* Wait before starting the playback again */
        R_BSP_SoftwareDelay(2, BSP_DELAY_UNITS_SECONDS);
    }
}

void g_transfer_dac_cb(dmac_callback_args_t * p_args)
{
    /* Use this callback to end the playback or restart the DMA
     * with more samples to play longer tracks */

    /* Signal that last sample has been sent to DAC */
    dac_done = true;

    /* Suppress compiler warning for unused p_args */
    FSP_PARAMETER_NOT_USED(p_args);
}


void R_BSP_WarmStart(bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_POST_C == event)
    {
        /* C runtime environment and system clocks are setup. */

        /* Configure pins. */
        R_IOPORT_Open (&g_ioport_ctrl, g_ioport.p_cfg);
    }
}
```

5.13

The example project provides **guitar.c** file which includes an example track stored as array of PCM inside const unsigned char audio_samples[130032]. You can replace this file with your own samples and/or buffer them in another array in on-chip SRAM. With DAC set to left-justified in step 5.6, the audio samples should be provided in unsigned 16-bit mono PCM format (regardless of whether the storage type in the code is 8, 16 or 32-bit). To convert any audio file to this format, use ffmpeg and execute the following:

```
ffmpeg.exe -i {input_file} -acodec pcm_u16le -f u16le -ac 1 -ar 16000 {output_file}
```

Where {input_file} and {output_file} are replaced by the path to input and output, respectively. “16000” after the “-ar” is the output sampling rate setting and should match timer rate set in step 5.5.

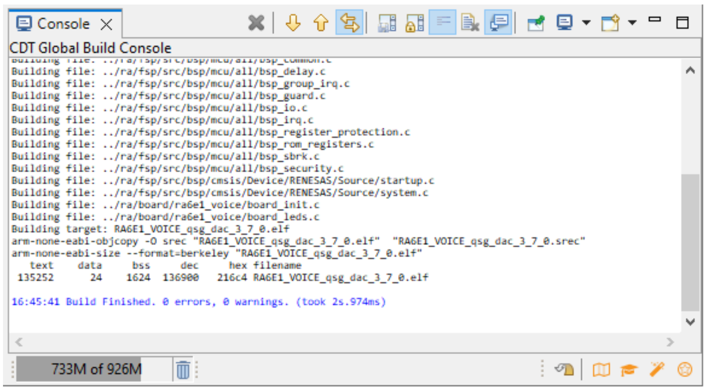
Raw audio files output by ffmpeg can be included in the project either by converting them to a C array or by creating an assembly file with .incbin directive to inline the file.

5.14

The project is now ready to compile. Press the “hammer” icon to start building the project.

5.15

Once the build has finished, the console pane in the lower-right corner of e^2 studio will report zero error and warnings:



```
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_common.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_delay.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_group_irq.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_guard.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_io.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_irq.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_register_protection.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_rom_registers.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_sbrk.c
Building file: ../ra/fsp/src/bsp/mcu/all/bsp_security.c
Building file: ../ra/fsp/src/bsp/cmsis/Device/RENESAS/Source/startup.c
Building file: ../ra/fsp/src/bsp/cmsis/Device/RENESAS/Source/system.c
Building file: ../ra/board/ra6e1_voice/board_init.c
Building file: ../ra/board/ra6e1_voice/board_leds.c
Building target: RA6E1_VOICE_qsg_dac_3_7_0.elf
arm-none-eabi-objcopy -O srec "RA6E1_VOICE_qsg_dac_3_7_0.elf"  "RA6E1_VOICE_qsg_dac_3_7_0.srec"
arm-none-eabi-size --format=berkeley "RA6E1_VOICE_qsg_dac_3_7_0.elf"
   text    data     bss     dec     hex filename
 135252      24    1624  136900   216c4 RA6E1_VOICE_qsg_dac_3_7_0.elf

16:45:41 Build Finished. 0 errors, 0 warnings. (took 2s.974ms)
```

5.16	The application is now ready to be programmed and run on the VOICE kit. Press the “bug” icon to begin the debug session.
5.17	You may be prompted to update the J-Link debugger firmware. You can click **Yes** to update. It will take a few moments to complete.
5.18	Windows could also prompt you to allow the GDB server through your firewall. Click the checkbox to allow it through private networks, then **Allow** access.
5.19	e^2 studio will perform flash programming routines and prompt to switch to **Debug** perspective. Select the check box by **Remember my decision** and click **Switch**.
5.20	The debug session is now started, and the application is paused at its entry function (`SystemInit()` in `Reset_Handler`). At this point, you can set up additional debug features such as variable and expressions views before the program is executed.
5.21	Click the **Resume** button or press **F8** on the keyboard to start the application.
5.22	The Program will stop again, this time at the start of the main function. Low-level initialization routines are now completed. Press **Resume** or **F8** again to resume the application and begin executing user code.
5.23	As application is executing, the green LED will toggle each time a new audio buffer is captured. The sound capture is running continuously with each new data set being passed to the main loop approximately every 500ms (16000Hz sampling rate with 8000 samples per buffer). The sample code implements double buffering to allow for further processing of the data without breaking the data continuity. Example application can be easily extended to use the data captured, e.g. for voice recognition model or real-time streaming to another host.
5.24	Click the **Terminate** button or press **Ctrl + F2** on the keyboard to stop the application and terminate the debug session.

6 Migrating projects from VOICE-RA6E1 kit

Overview

Following section explains how to migrate projects originally created for VOICE-RA6E1 kit to run on VOICE-RA4E1 kit.

Procedural Steps

All VOICE kit designs attempt to maintain consistent layout and pin mapping between different variants. As such, projects using various features on VOICE-RA6E1 can be easily migrated to run on VOICE-RA4E1, provided RAM and ROM requirements to run the project are satisfied.

For all projects, navigate to the BSP tab in the FSP Configuration and select VOICE-RA4E1 board.

Based on peripherals used, following additional changes need to be made:

- For projects using UART driver for USB-to-UART functionality:
    - In the properties for UART instance, change General -> **Channel from 4 to 3**
- For projects using digital microphones:
    - In the properties for GPT instance used for I2S SCK signal, change Output -> **GTIOCA Output Enable from Ture to False**; change Output -> **GTIOCB Output Enable from False to True**.
- For projects using analog microphones: no changes are needed.
- For projects using audio output through DAC: no changes are needed.
- For projects using PMOD: consult the board manual or schematic to obtain new pin assignments.
- For projects using QSPI: no changes are needed, QSPI is always enabled on the VOICE-RA4E1 kit. Expect QSPI throughput to be roughly halved compared to VOICE-RA6E1, as it is configured in Dual I/O mode on VOICE-RA4E1.

END OF THE QUICK SETUP GUIDE