Quick Setup Guide for VOICE-RA2L1 VUI Solution Kit

Renesas Advanced (RA) Family – RA2 Series

Description

Welcome to a Quick Setup Guide for VOICE-RA2L1 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-RA2L1
• Enable serial terminal output
• Enable digital microphones
• Enable analog microphones
• Enable audio playback
	Prerequisites
	• Renesas VOICE-RA2L1 VUI Solution Kit
	• Renesas Flexible Software Package platform installation, which includes:
	  • e^2 studio 2022-04 or newer
	  • FSP 4.0.0 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 2

1 Installing BSP and creating an FSP project 3

2 Configuring and using serial communications 6

3 Configuring and using digital microphones 12

4 Configuring and using analog microphones 18

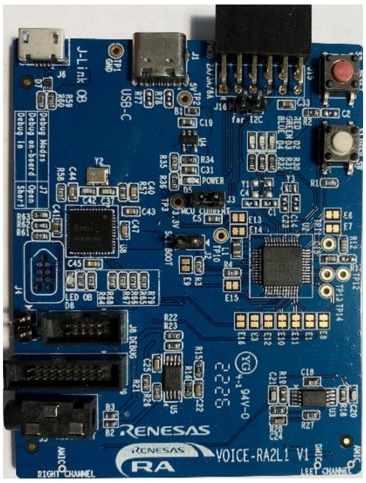
5 Configuring and using audio output 23

0 Setting up the hardware

Overview

Following section describes in details steps required to create an e^2 studio workspace and set up a project for RA2L1-VOICE kit

Procedural Steps

0.1	To begin working with VOICE-RA2L1 platform you will need the following:  VOICE-RA2L1 VUI Solution Kit  USB micro-B cable (included with the kit)
0.2	Connect the VOICE-RA2L1 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.

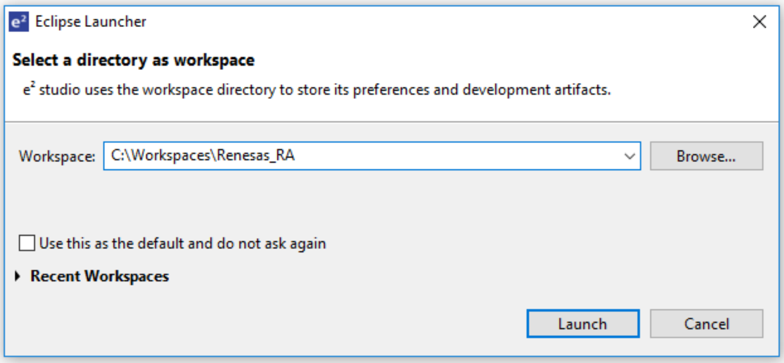
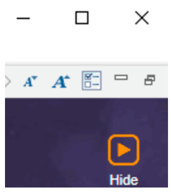
END OF SECTION

1 Installing BSP and creating an FSP project

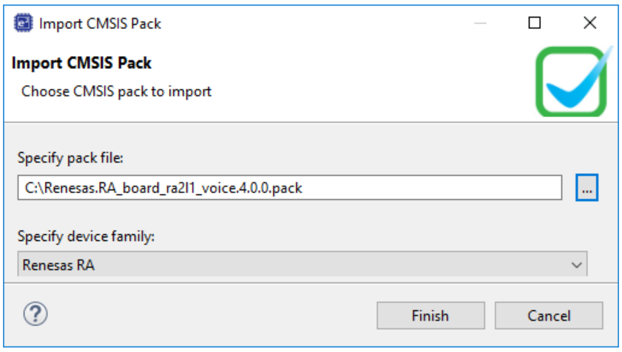
Overview

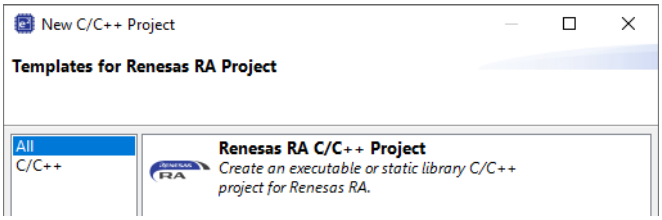
Following section describes in details steps required to create an e² studio workspace and set up a project for RA2L1-VOICE kit.

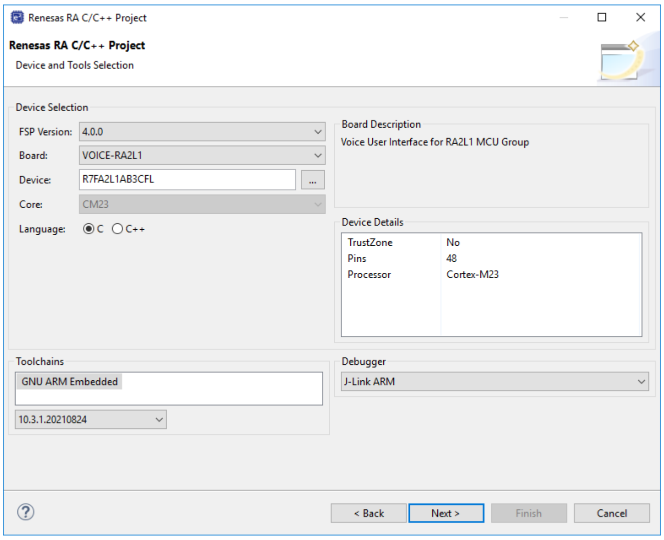
Procedural Steps

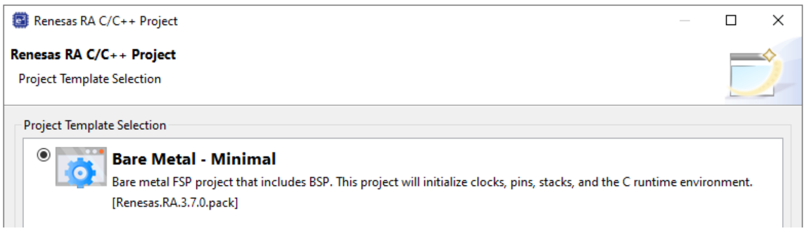
1.1	Launch e² studio. e² studio can be launched from the Windows start menu or directly from the installation folder. If you have multiple versions of e² studio installed, please make sure to launch the version of e² studio that was specified on the first page.
1.2	In the Eclipse Launcher window, specify the destination for the new workspace. It is recommended to keep the path simple and avoid using spaces. 
1.3	Click **Launch** to start e² 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**.

<table>
<tr><td>1.6</td><td>In the Import CMSIS Pack window, click … to browse for the .pack file containing BSP for VOICE-RA2L1 kit (Renesas.RA_board_RA2L1_voice.<version>.pack). Select Renesas RA from the drop-down box under Specify device family and click Finish.


</td></tr>
<tr><td>1.7</td><td>Click OK in the pop-up window confirming successful pack file import.</td></tr>
<tr><td>1.8</td><td>Go to File -> New and select Renesas C/C++ Project, then Renesas RA.</td></tr>
<tr><td>1.9</td><td>In the new project wizard window, select Renesas RA C/C++ Project and click Next.


</td></tr>
<tr><td>1.10</td><td>Specify a project name and Click Next.</td></tr>
<tr><td>1.11</td><td>Select FSP version matching your BSP and FSP installation (e.g., 4.0.0) and set Board to VOICE-RA2L1. Verify that the Debugger is set to J-Link ARM and click Next.


</td></tr>
<tr><td>1.12</td><td>Select Flat (Non-TrustZone) Project and click Next.</td></tr>
<tr><td>1.13</td><td>On the next window, leave Executable and No RTOS selected. Click Next.</td></tr>
</table>

1.14	On the final page of the new project wizard select **Bare Metal – Minimal** and click **Finish**. 
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.

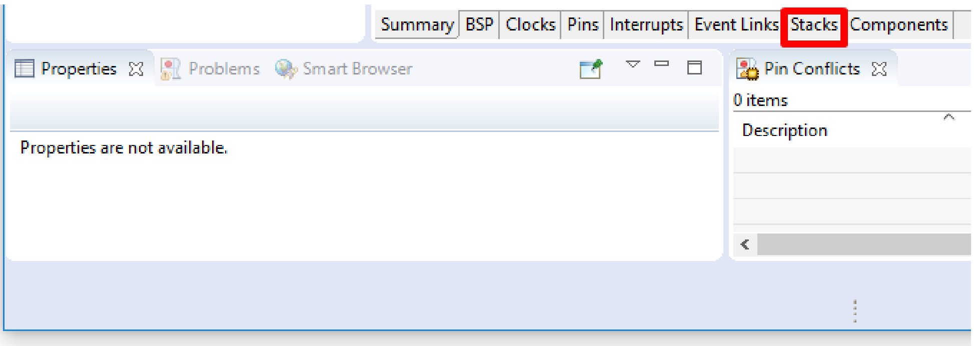
2 Configuring and using serial communications

Overview

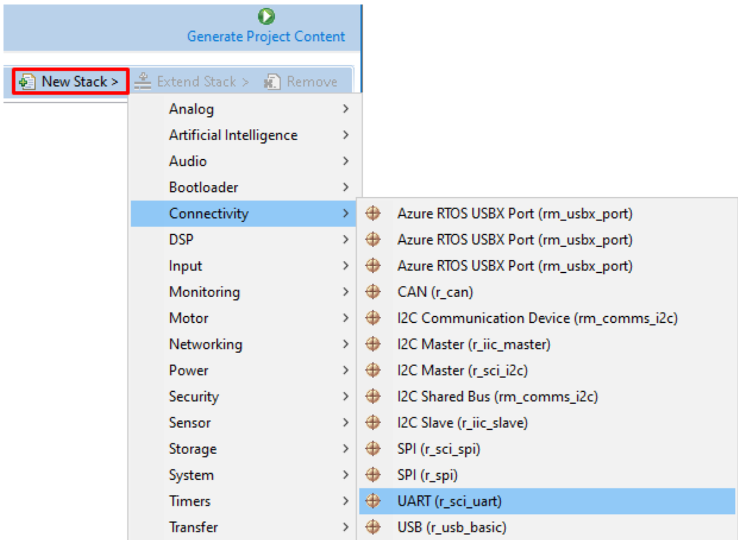
Following section explains how to configure and operate basic UART write and read functionality on the VOICE kit.

Procedural Steps

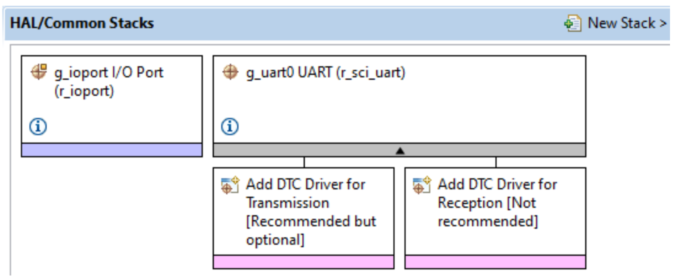
<table>
<tr><td>2.1</td><td>This section begins with an empty RA2L1-VOICE project. Make sure steps 1.8-1.15 are completed before starting this section.</td></tr>
<tr><td>2.2</td><td>Once new project is created, e^2 studio will switch to a layout optimized for developing Renesas RA projects. Select the Stacks tab at the bottom of the FSP Configuration pane visible in the middle.



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

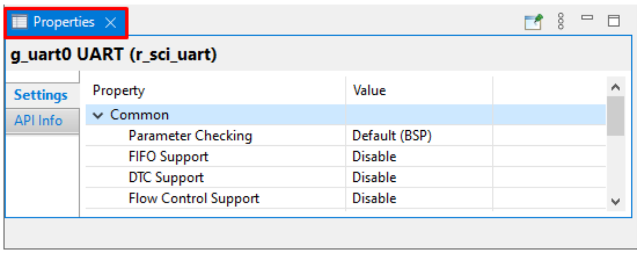


</td></tr>
<tr><td>2.4</td><td>A new driver module will be added inside the HAL/Common group.



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

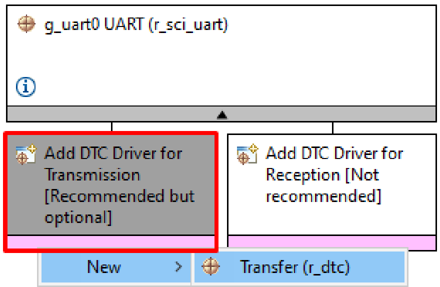
2.5 Click on **g_uart0 UART (r_sci_uart)** and go to the **Properties** tab. It can be found in the lower-left pane, directly under the **Project Explorer**.



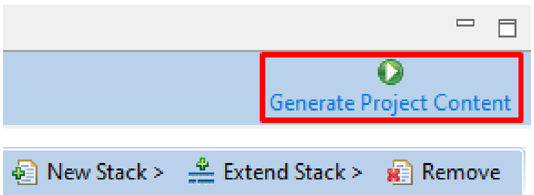
2.6 Set the following properties for g_uart0. You may need to expand the chevrons to access all of the properties:

- 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.

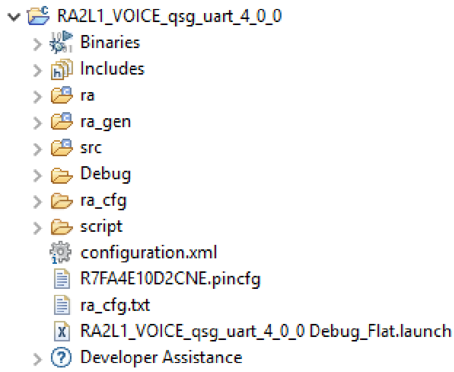


2.8 RA Configuration for this section is complete. Apply changes to the project source by clicking the **Generate Project Content** button in the top-right corner of the Configurator window. When prompted to *Proceed with save and generate*, tick the box next to **Always save and generate without asking** and click **Proceed**.



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**.



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).

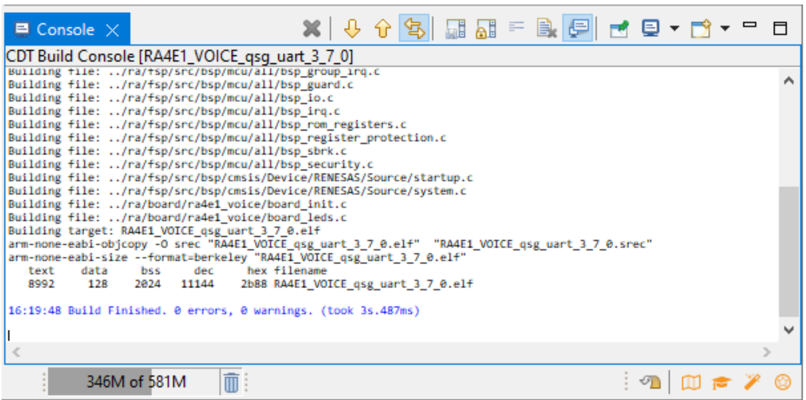
2.12 **hal_entry.c** can be used to exercise API of the various modules configured inside FSP Configurator using Developer Assist or by writing code manually. Following code can be used to completely replace contents of hal_entry.c to perform basic UART write and read operations:

```
#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 */
```

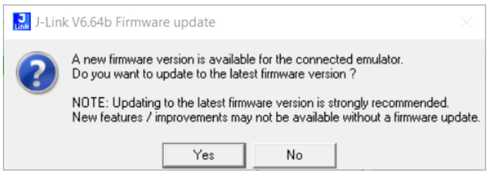
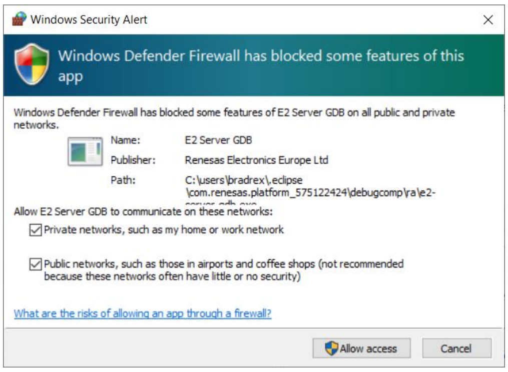
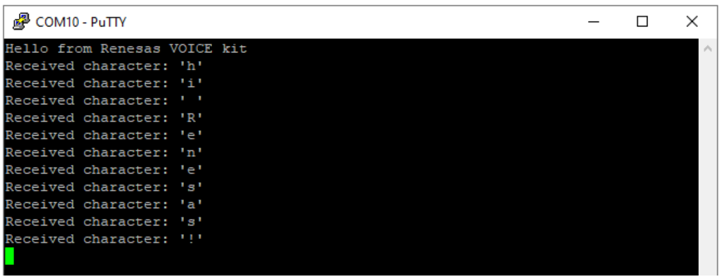
```
        err = R_SCI_UART_Write(&g_uart0_ctrl, (void *) text_buf, strlen(text_buf));
        if (FSP_SUCCESS != err)
        {
            __BKPT(0);
        }

        /* 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);
    }
}
```

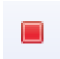
2.13 The project is now ready to compile. Press the “hammer” icon to start building the project.

2.14 Once the build has finished, the console pane in the lower-right corner of e² studio will report zero error and warnings:



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.

<table>
<tr><td>2.16</td><td>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.

</td></tr>
<tr><td>2.17</td><td>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.

</td></tr>
<tr><td>2.18</td><td>e² studio will perform flash programming routines and prompt to switch to Debug perspective. Select the check box by Remember my decision and click Switch.</td></tr>
<tr><td>2.19</td><td>The debug session is now started, and the application is paused at its entry function (<code>SystemInit()</code> in <code>Reset_Handler</code>). At this point, you can set up additional debug features such as variable and expressions views before the program is executed.</td></tr>
<tr><td>2.20</td><td>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.</td></tr>
<tr><td>2.21</td><td>Click the Resume button or press F8 on the keyboard to start the application.
</td></tr>
<tr><td>2.22</td><td>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.</td></tr>
<tr><td>2.23</td><td>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):

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

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

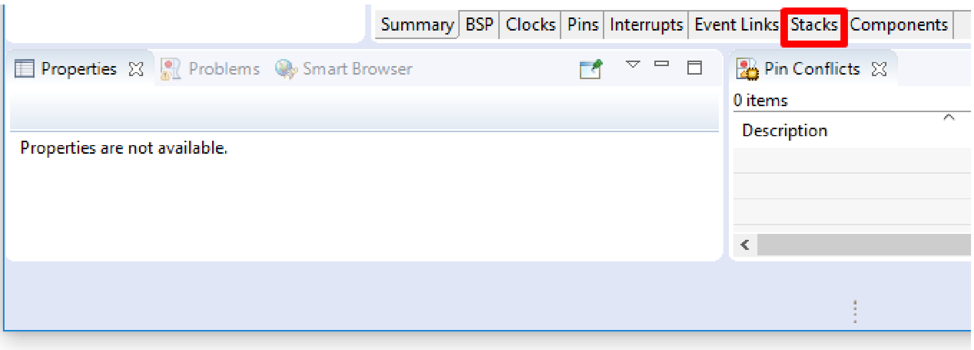
3 Configuring and using digital microphones

Overview

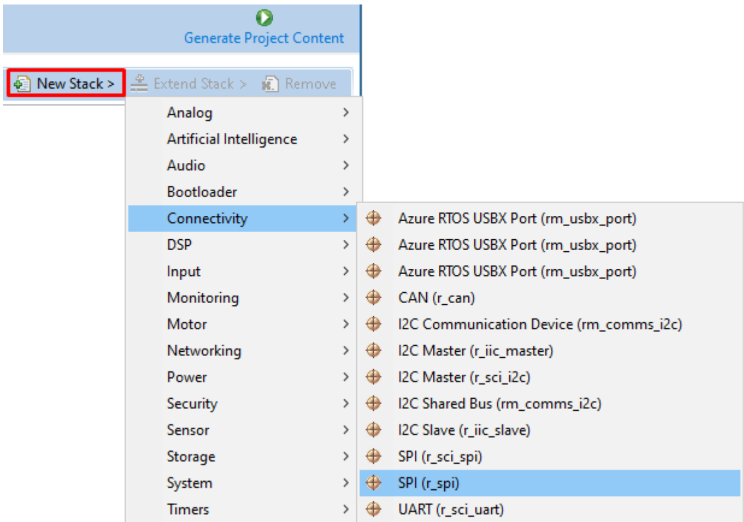
Following section explains how to configure and operate a digital microphone to capture audio input on the VOICE kit.

Procedural Steps

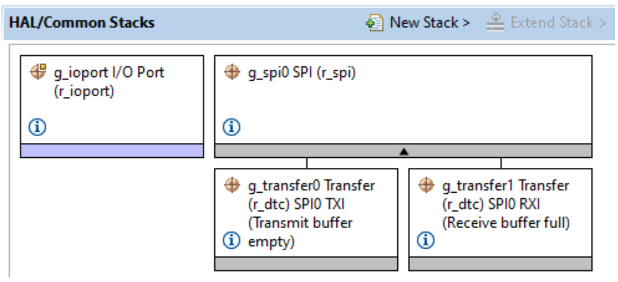
<table>
<tr><td>3.1</td><td>This section begins with an empty RA2L1-VOICE project. Make sure steps 1.8-1.15 are completed before starting this section.</td></tr>
<tr><td>3.2</td><td>Once new project is created, e[2] studio will switch to a layout optimized for developing Renesas RA projects. Select the Stacks tab at the bottom of the FSP Configuration pane visible in the middle.



</td></tr>
<tr><td>3.3</td><td>Open the New Stack menu (near the top-left corner) and navigate to Connectivity -> SPI (r_spi).



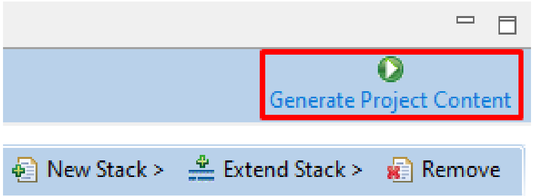
</td></tr>
<tr><td>3.4</td><td>A new driver module will be added inside the HAL/Common group. Unlike some other drivers (e.g. UART), DTC modules will be populated automatically.



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

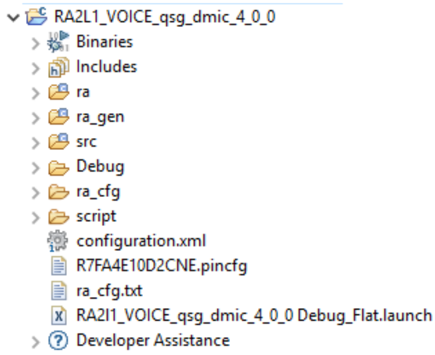
3.5	Click on **g_spi0 SPI (r_spi)**, go to the **Properties** tab and apply the following settings. You may need to expand the chevrons to access all of the properties: • Operating Mode — Slave • Callback — g_spi0_cb • SPI Mode — SPI Operation
3.6	Access the **New Stack** menu again and select **Timers -> 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_sclk • General -> Channel — 5 • General -> Period — 1024000 • General -> Period Unit — Hertz • Output -> GTIOC**B** 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 — 6 • General -> Mode — PWM • General -> Period — 32 • General -> Period Unit — Raw Counts • Output -> Custom Waveform -> GTIO**B** -> Initial Output Level — Pin Level High • Output -> Custom Waveform -> GTIO**B** -> Compare Match Output Level — Pin Level Toggle • Output -> Custom Waveform -> Custom Waveform Enable — Enabled • Output -> GTIOC**B** Output Enabled — True • Input -> Count Up Source — GTETRGB Failing Edge (check the box).
3.8	Access the **New Stack** menu and navigate to **Timers -> Port Output Enable for GPT (r_poeg)**. Use **Properties** tab to configure following properties for this new module: • General -> Name — g_poeg_ws • General -> Channel — 1
3.9	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_ssl • General -> Channel — 3 • General -> Mode — PWM • General -> Period — 32 • General -> Period Unit — Raw Counts • Output -> Custom Waveform -> GTIO**B** -> Initial Output Level — Pin Level High • Output -> Custom Waveform -> GTIO**B** -> Compare Match Output Level — Pin Level Toggle • Output -> Custom Waveform -> Custom Waveform Enable — Enabled • Output -> GTIOC**B** Output Enabled — True • Input -> Count Up Source — GTETRGB Failing Edge (check the box).

3.10 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**.



3.11 The FSP Configurator will extract all the necessary drivers and generate the code based on the configuration provided in the **Properties** tab.

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



3.13 **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).

3.14 **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:

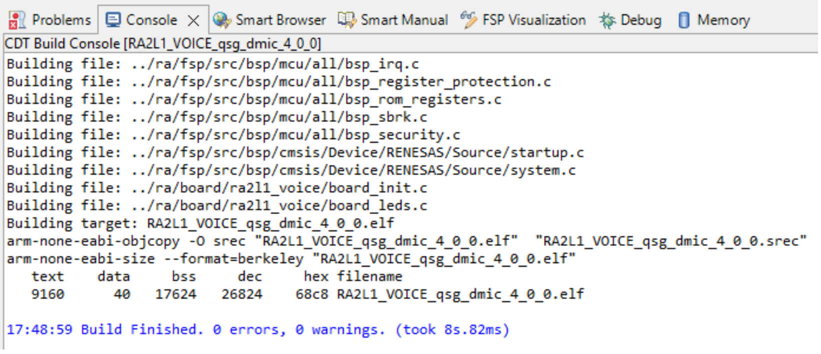
```c
#include "hal_data.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER

#define DMIC_BUF_SIZE   (2000)

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;

    /* Open timer used for I2S BCLK output */
    err = R_GPT_Open(&g_timer_sck_ctrl, &g_timer_sck_cfg);
    if(FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    err = R_POEG_Open(&g_poeg_ws_ctrl, &g_poeg_ws_cfg);
    if (FSP_SUCCESS != err)
```

```
    {
       __BKPT(0);
    }

    /* Open timer used for I2S WS output */
    err = R_GPT_Open(&g_timer_ws_ctrl, &g_timer_ws_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Delay slave select output by 1 clock wrt. I2S WS */
    err = R_GPT_CounterSet(&g_timer_ws_ctrl, g_timer_ws_cfg.period_counts - 13);
    if(FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    err = R_GPT_Open(&g_timer_ssl_ctrl, &g_timer_ssl_cfg);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    err = R_GPT_CounterSet(&g_timer_ssl_ctrl, g_timer_ssl_cfg.period_counts - 14);
    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Start WS and SS timers */
    err = R_GPT_Start(&g_timer_ws_ctrl);
    if(FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    err = R_GPT_Start(&g_timer_ssl_ctrl);
    if(FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Initialize SPI peripheral */
    err = R_SPI_Open(&g_spi_i2s_ctrl, &g_spi_i2s_cfg);
    if(FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    err = R_GPT_Start(&g_timer_sck_ctrl);
    if(FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Set up the initial I2S read */
    err = R_SPI_Read(&g_spi_i2s_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_spi_i2s_ctrl);
            R_SPI_Open(&g_spi_i2s_ctrl, &g_spi_i2s_cfg);
            do
            {
                /* Repeat this request if it fails */
                err = R_SPI_Read(&g_spi_i2s_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_spi_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_spi_i2s_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);
    }
}
```

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

<table>
<tr><td>3.16</td><td>Once the build has finished, the console pane in the lower-right corner of e² studio will report zero error and warnings:


</td></tr>
<tr><td>3.17</td><td>The application is now ready to be programmed and run on the VOICE kit. Press the “bug” icon to begin the debug session.

</td></tr>
<tr><td>3.18</td><td>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.</td></tr>
<tr><td>3.19</td><td>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.</td></tr>
<tr><td>3.20</td><td>e² studio will perform flash programming routines and prompt to switch to Debug perspective. Select the check box by Remember my decision and click Switch.</td></tr>
<tr><td>3.21</td><td>The debug session is now started, and the application is paused at its entry function (<code>SystemInit()</code> in <code>Reset_Handler</code>). At this point, you can set up additional debug features such as variable and expressions views before the program is executed.</td></tr>
<tr><td>3.22</td><td>Click the Resume button or press F8 on the keyboard to start the application.

</td></tr>
<tr><td>3.23</td><td>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.</td></tr>
<tr><td>3.24</td><td>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.</td></tr>
<tr><td>3.25</td><td>Click the Terminate button or press Ctrl + F2 on the keyboard to stop the application and terminate the debug session.

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

END OF SECTION

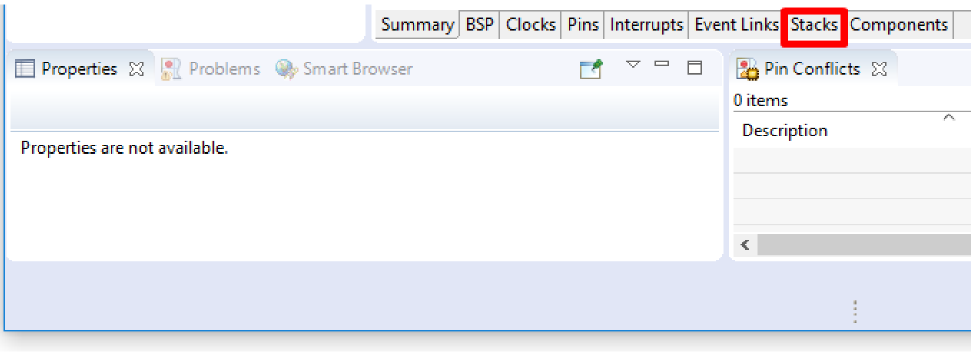
4 Configuring and using analog microphones

Overview

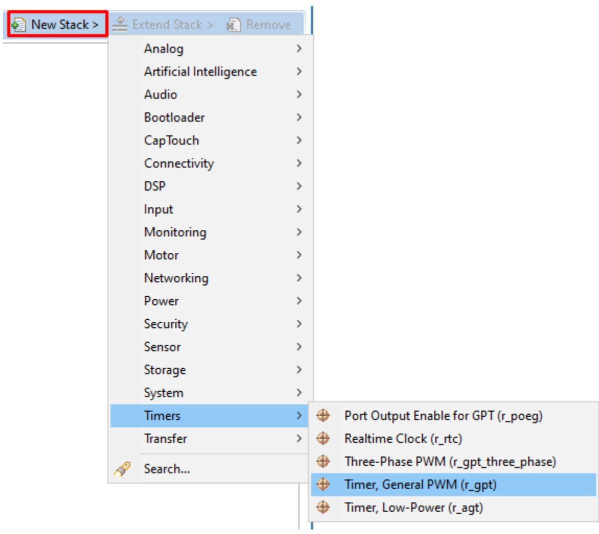
Following section explains how to configure and operate a pair of analog microphones to capture audio input on the VOICE kit.

Procedural Steps

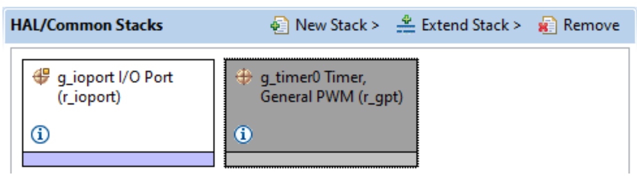
<table>
<tr><td>4.1</td><td>This section begins with an empty RA2L1-VOICE project. Make sure steps 1.8-1.15 are completed before starting this section.</td></tr>
<tr><td>4.2</td><td>Once new project is created, e^2 studio will switch to a layout optimized for developing Renesas RA projects. Select the Stacks tab at the bottom of the FSP Configuration pane visible in the middle.



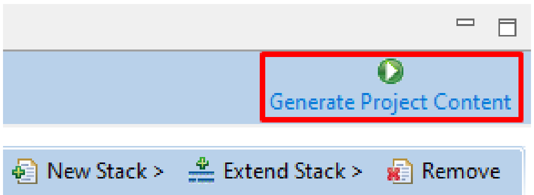
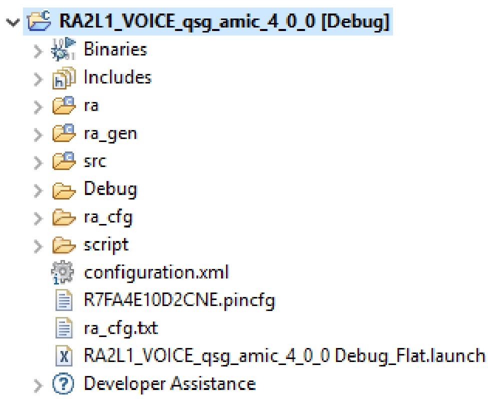
</td></tr>
<tr><td>4.3</td><td>Open the New Stack menu and navigate to Timers -> Timer, General PWM (r_gpt).



</td></tr>
<tr><td>4.4</td><td>A new driver module will be added inside the HAL/Common group.



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

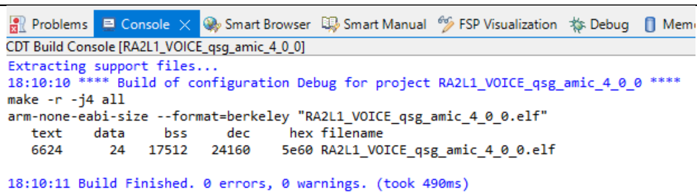
4.5	Click on **g_timer0 Timer, General PWM (r_gpt)**, go to the **Properties** tab and apply the following settings. You may need to expand the chevrons to access all of the properties: • General -> Channel 2 • General -> Period 16000 • General -> Period Unit Hertz
4.6	Access the **New Stack** menu again 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 -> Data Transfer Controller (DTC)**. Use **Properties** tab to configure following properties for this new module: • Name g_transfer0 • Transfer Size 4 Bytes • Destination Address Mode Incremented • Activation Source ADC0 SCAN END • Callback g_transfer0_cb
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**. 
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**. 

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   (2000)

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 DCT peripheral */
    //err = R_DMAC_Open(&g_transfer0_ctrl, &g_transfer0_cfg);
    err = R_DTC_Open(&g_transfer0_ctrl, &g_transfer0_cfg);

    if (FSP_SUCCESS != err)
    {
        __BKPT(0);
    }

    /* Set the DCT to capture from ADC registers into amic_buf */
    err = R_DTC_Reset(&g_transfer0_ctrl, (void *) R_ADC0->ADDR, amic_buf[amic_idx],
AMIC_BUF_SIZE);
```

```c
    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(adc_callback_args_t * p_args)
{
    /* Change index of the active write buffer */
    amic_idx ^= 1;

    /* Start subsequent ADC capture */
    R_DTC_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:

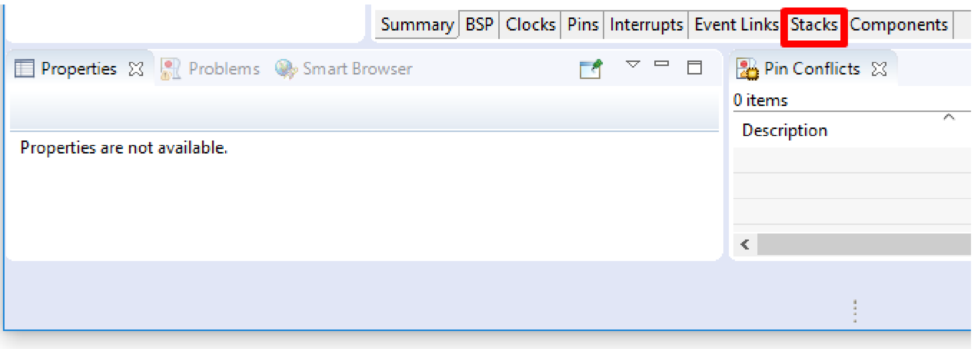
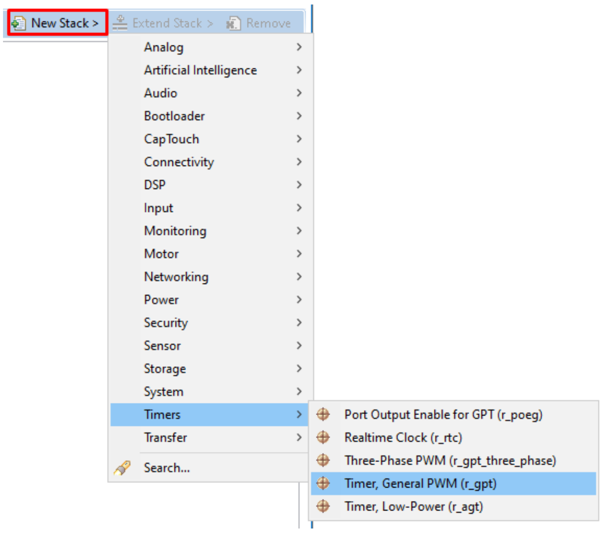
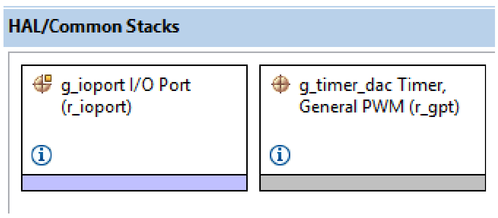
	
4.16	The application is now ready to be programmed and run on the VOICE kit. Press the “bug” icon to begin the debug session. 
4.17	You may be prompted to update the J-Link debugger firmware. You can click **Yes** to update. It will take a few moments to complete.
4.18	Windows could also prompt you to 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 RA2L1-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. 
5.3	Open the **New Stack** menu and navigate to **Timers -> Timer, General PWM (r_gpt)**. 
5.4	A new driver module will be added inside the **HAL/Common group**. 

5.5

Click on **g_timer_dac 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

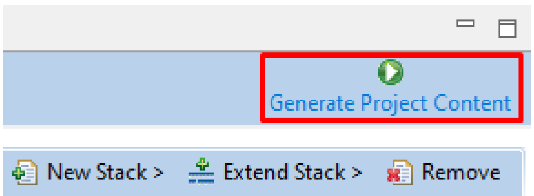
Access the **New Stack** menu yet again and select **Transfer -> Data Transfer Controller (DTC)**. Use **Properties** tab to configure following properties for this new module:

- Name   g_transfer_dac
- Transfer Size   2 Bytes
- Destination Address Mode   Incremented
- Activation Source   GPT5 COUNTER OVERFLOW

* 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**.

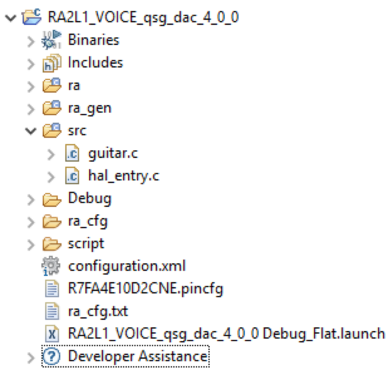


5.9

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**.



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:

```c
#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 DTC peripheral */
    err = R_DTC_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_DTC_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;
```

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

```c
        /* Wait before starting the playback again */
        R_BSP_SoftwareDelay(2, BSP_DELAY_UNITS_SECONDS);
    }
}

void g_transfer_dac_cb(timer_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);
    }
}
```

</td></tr>
<tr><td>5.13</td><td>

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.

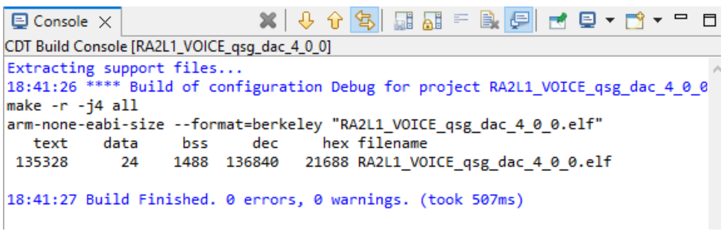
Raw audio files output by ffmpeg can be included in the project either by converting them to a C array or by creating an assembly file with .incbin directive to inline the file.

</td></tr>
<tr><td>5.14</td><td>

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

</td></tr>
<tr><td>5.15</td><td>

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



</td></tr>
<tr><td>5.16</td><td>

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

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

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. 

END OF THE QUICK SETUP GUIDE