Quick Setup Example on AIK-RA8D1 Solution Kit

Renesas Advanced (RA) Family – RA8 Series

Description

Welcome to Quick Setup Example for Renesas RA using AIK-RA8D1 Solution Kit! The objective of this workshop is to build a basic Renesas RA application utilizing Renesas tools.

You will start by setting up the display with the basic operations project. The application used in this lab is built to run on AIK-RA8D1 Solution Kit. A foundation Display project will be created from scratch and populated with several HAL drivers provided by the Flexible Software Package (FSP). Accelerometer and Ethernet demo projects are also added.

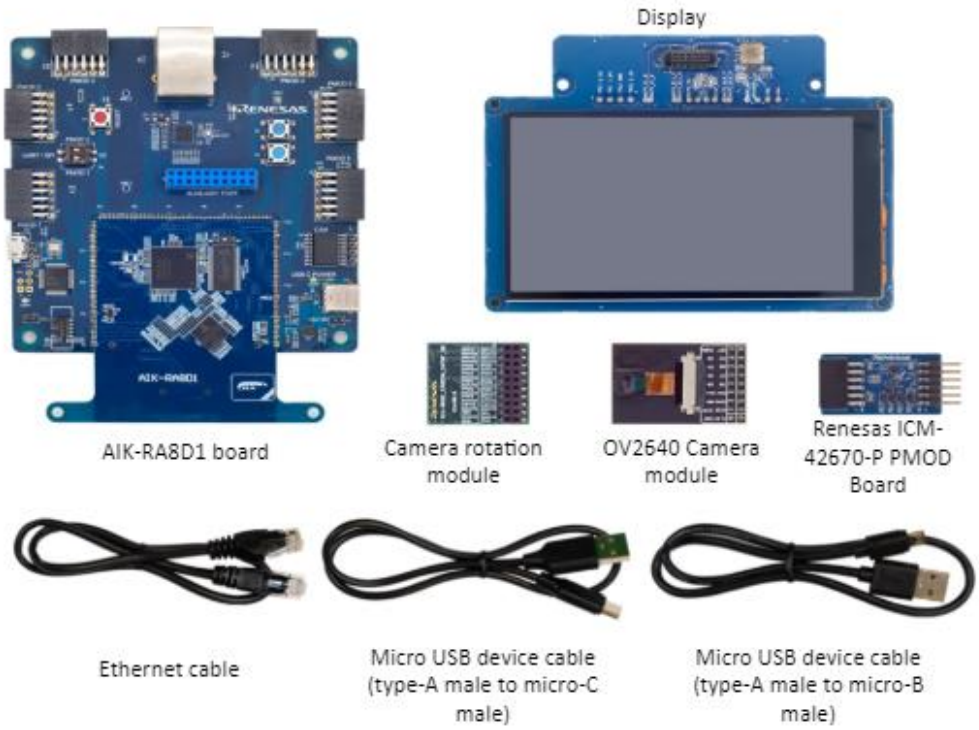
Objectives
• Configure AIK-RA8D1 kit to run display with the basic operations project
• Implement Accelerometer demo
• Implement Ethernet demo
	Prerequisites
	• Renesas AIK-RA8D1 VUI Solution Kit
	• Renesas Flexible Software Package 5.5.0 platform installation, which includes:
	  • e² studio 2024-07 or newer
	  • FSP 5.5.0 or newer
	  • GCC Arm Embedded 13.2.1
	• PC running Windows 10 64-bit with at least one USB port.
	• Serial terminal software such as PuTTY or TeraTerm (provided with the workshop)
	• J-Link RTT Viewer
	• Router with Ethernet connection

Skill Level • Basic familiarity with embedded electronics • Basic understanding of C language • Understanding of how to import projects into e² studio (optional – for use with ready checkpoint projects).	Time • 2 hours to complete

Workshop Sections

0 Setting up the hardware

Procedural Steps

0.1	To begin working with AIK-RA8D1 platform you will need the following: 
0.2	Connect the AIK-RA8D1 kit to the PC using USB micro-B cable and J-Link OB USB port (J10) in the bottom left corner of the board. Connect Renesas ICM-42670-P PMOD Board to PMOD1.
0.3	Verify that: • The blue LED2 (POWER) near the ethernet port is on. • The orange LED4 (LED OB) 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.

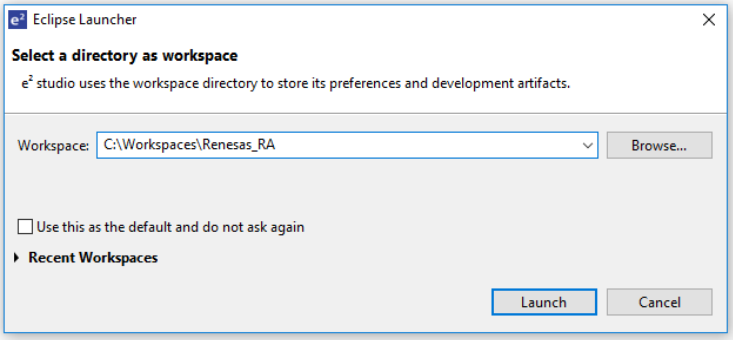
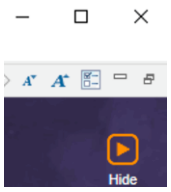
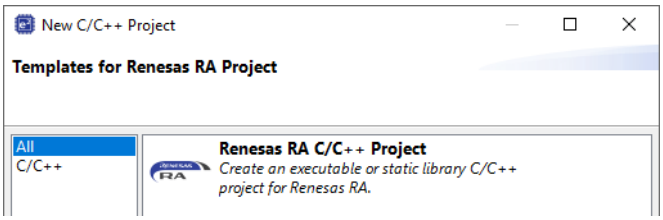
END OF SECTION

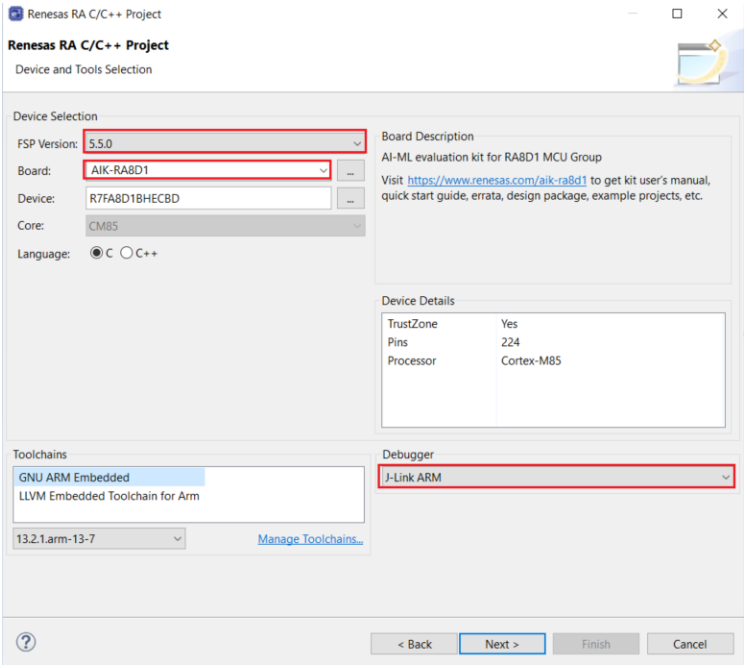
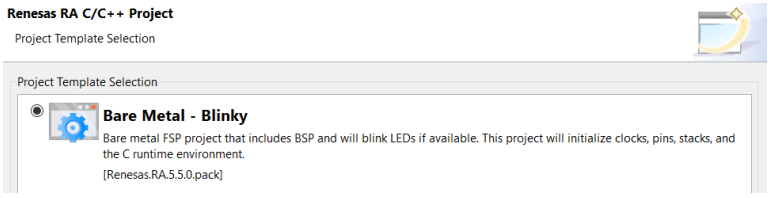
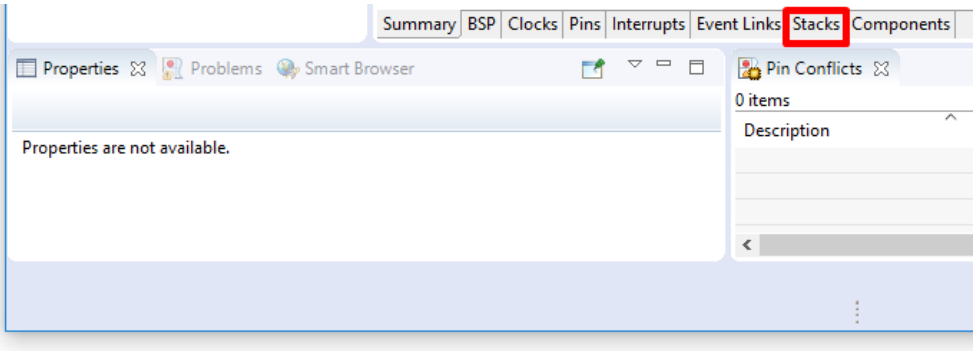
1 Implementing Display with the basic operations demo

Overview

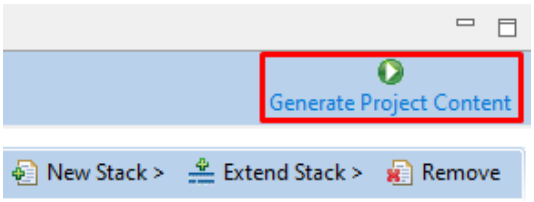
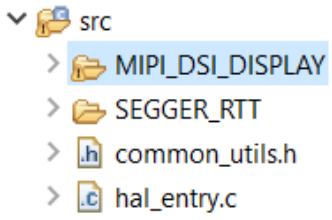
Following section describes in details steps required to create an e² studio workspace and set up a Display with basic operations-based project for AIK RA8D1 kit.

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 the FSP setup that is aimed to be used. The FSP migration guide found in renesas/aiot-ready (same as this document path) it will help you to switch between FSP versions without facing compatibility or other issues in the development environment.
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 -> New** and select **Renesas C/C++ Project**, then **Renesas RA**.
1.6	In the new project wizard window, select **Renesas RA C/C++ Project** and click **Next**. 
1.7	Specify a project name and Click **Next**.

1.8	Select FSP version matching your BSP and FSP installation (e.g., **5.5.0**) and set Board to **AIK-RA8D1**. Verify that the Debugger is set to **J-Link ARM** and click **Next**. 
1.9	On the next window, leave **Executable** and **No RTOS** selected. Click **Next**.
1.10	On the final page of the new project wizard select **Bare Metal – Blinky** and click **Finish**. 
1.11	When prompted to open the **FSP Configuration perspective**, click **Open Perspective**. The project is now set up to begin evaluation and development using the AIK kit.
1.12	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. 

1.13	Access the **New Stack** menu again and select **Input -> External IRQ (r_icu)**. Use **Properties** tab to configure following properties for this new module: • Name g_external_irq • Channel 13 • Trigger Rising • Digital Filtering Enable • Callback external_irq_callback • Overflow/Crest Interrupt Priority Priority 1
1.14	Open the **New Stack** menu (near the top-left corner) and navigate to **Timers -> Timer, General PWM (r_gpt)**. Use **Properties** tab to configure following properties for this new module: • Pin Output Support Enabled • Name g_timer0 • Channel 0 • Mode Periodic • Period 0x10000 • Period Unit Raw Counts • Callback gpt_callback
1.15	Open the **New Stack** menu (near the top-left corner) and navigate to **Connectivity -> I2C Master (r_iic_master)**. Use **Properties** tab to configure following properties for this new module: • Name g_i2c_master • Channel 1 • Rate Standard • Rise Time 120 • Fall Time 120 • Duty Cycle 50 • Slave Address 0x14 • Callback i2c_master_callback
1.16	Navigate to **Graphics** on **Graphics LCD (r_glcdc)**, go to the **Properties** tab and apply the following settings. You may need to expand the chevrons to access all of the properties: • Name g_display • Callback glcdc_callback • Line Detect Interrupt Priority Priority 1 • Input—Graphics Layer 1 –Color format RGB888(32-bit) Click and add **Add Mipi DSI Output** and in properties change: • Callback mipi_dsi_callback

1.17	Open the **New Stack** menu (near the top-left corner) and navigate to **Timers -> Timer, General PWM (r_gpt)**. Use **Properties** tab to configure following properties for this new module: • Pin Output Support Enabled • Name g_timer_pwm_backlight_1 • Channel 1 • Mode Periodic • Period 22 • Period Unit Kilohertz • GTIOCB Output Enabled True
1.18	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**. 
1.19	The FSP Configurator will extract all the necessary drivers and generate the code based on the configuration provided in the **Properties** tab.
1.20	In the **Project Explorer** pane, expand the **src** folder in the project and add the following folders and files that can be found in the demo folder: • MIPI_DSI_DISPLAY • SEGGER_RTT • common_utils.h
1.21	In the **Project Explorer** pane, expand the **src** folder in the project and open **hal_entry.c**. 
1.22	**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).
1.23	Add #include statement to include the following near the top of the file. #include <MIPI_DSI_DISPLAY/mipi_dsi_ep.h> #include "hal_data.h" #include "r_mipi_dsi.h" #include "common_utils.h"

1.24

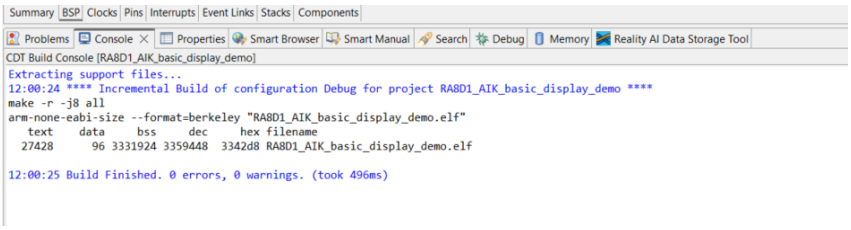
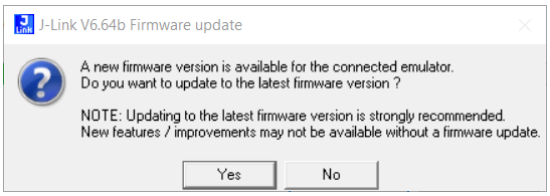
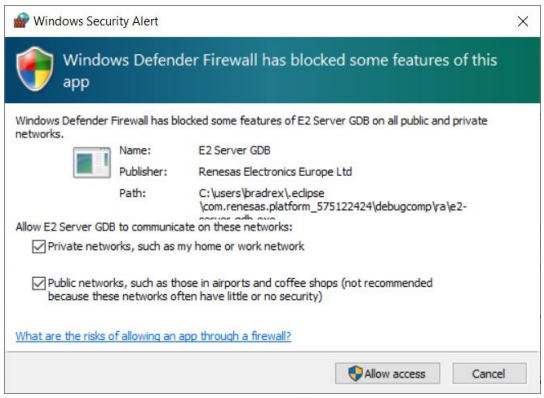
hal_entry.c can be used to exercise API of the various modules configured inside FSP Configurator using Developer Assist or by writing code manually.

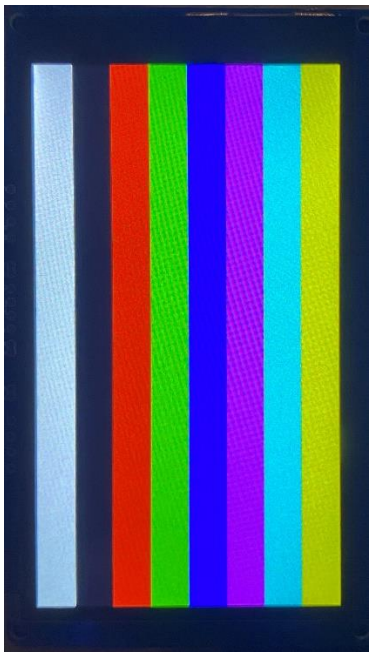
Following code can be used to completely replace contents of hal_entry.c to perform basic operations using the display for the AIK-RA8D1 board:

```
#include <MIPI_DSI_DISPLAY/mipi_dsi_ep.h>
#include "hal_data.h"
#include "r_mipi_dsi.h"
#include "common_utils.h"

void R_BSP_WarmStart(bsp_warm_start_event_t event);

extern bsp_leds_t g_bsp_leds;

/*******************************************************************************************************************//**
 * @brief  Blinky example application
 *
 * Blinks all leds at a rate of 1 second using the software delay function provided by the BSP.
 *
 **********************************************************************************************************************/
void hal_entry (void)
{
#if BSP_TZ_SECURE_BUILD

    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif

    /* Define the units to be used with the software delay function */
    const bsp_delay_units_t bsp_delay_units = BSP_DELAY_UNITS_MILLISECONDS;

    /* Set the blink frequency (must be <= bsp_delay_units */
    const uint32_t freq_in_hz = 2;

    /* Calculate the delay in terms of bsp_delay_units */
    const uint32_t delay = bsp_delay_units / freq_in_hz;

    /* LED type structure */
    bsp_leds_t leds = g_bsp_leds;

    /* If this board has no LEDs then trap here */
    if (0 == leds.led_count)
    {
        while (1)
        {
            ;                           // There are no LEDs on this board
        }
    }

    /* Holds level to set for pins */
    bsp_io_level_t pin_level = BSP_IO_LEVEL_LOW;

    while (1)
    {
        /* Enable access to the PFS registers. If using r_ioport module then register
protection is automatically
         * handled. This code uses BSP IO functions to show how it is used.
         */
        R_BSP_PinAccessEnable();

        /* Update all board LEDs */
        for (uint32_t i = 0; i < leds.led_count; i++)
        {
            /* Get pin to toggle */
            uint32_t pin = leds.p_leds[i];

            /* Write to this pin */
            R_BSP_PinWrite((bsp_io_port_pin_t) pin, pin_level);
        }
```

```
        /* Protect PFS registers */
        R_BSP_PinAccessDisable();

        /* Toggle level for next write */
        if (BSP_IO_LEVEL_LOW == pin_level)
        {
            pin_level = BSP_IO_LEVEL_HIGH;
        }
        else
        {
            pin_level = BSP_IO_LEVEL_LOW;
        }

        /* Delay */
        R_BSP_SoftwareDelay(delay, bsp_delay_units);

        mipi_dsi_entry();

    }
}

/*******************************************************************************************************************//**
 * This function is called at various points during the startup process.  This implementation uses the event that is
 * called right before main() to set up the pins.
 *
 * @param[in]  event    Where at in the start up process the code is currently at
 **********************************************************************************************************************/
void R_BSP_WarmStart (bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_RESET == event)
    {
#if BSP_FEATURE_FLASH_LP_VERSION != 0

        /* Enable reading from data flash. */
        R_FACI_LP->DFLCTL = 1U;

        /* Would normally have to wait tDSTOP(6us) for data flash recovery. Placing the enable here, before clock and
         * C runtime initialization, should negate the need for a delay since the initialization will typically take more than 6us. */
#endif
    }

    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);
    }
}
```

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

1.26	Once the build has finished, the **Console** pane in the lower-right corner of e² studio will report zero errors : 
1.27	The application is now ready to be programmed and run on the AIK kit. Press the “bug” icon to begin the debug session. 
1.28	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. 
1.29	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. 
1.30	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**.
1.31	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.
1.32	Click the Resume button or press F8 on the keyboard to start the application. 
1.33	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.

1.34	Go back to the devices display to observe the output from the AIK kit. The display will show Multiple layers with different colors. 
1.35	Click the **Terminate** button or press **Ctrl + F2** on the keyboard to stop the application and terminate the debug session. 
1.36	Full project is also available in GitHub under https://github.com/renesas/aiot-ready/tree/master/kits/RA8D1_aik/qse

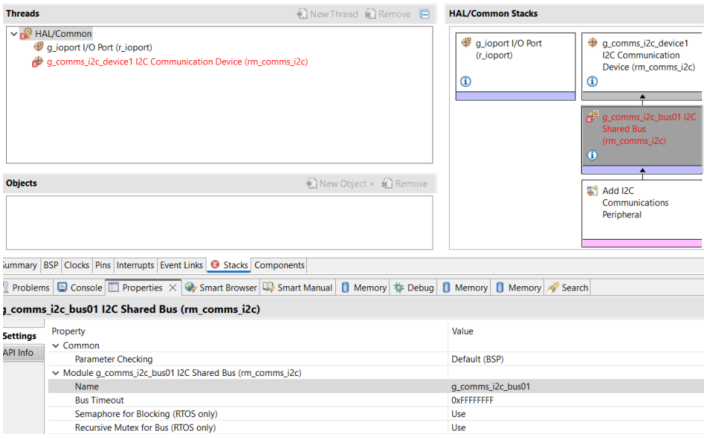
END OF SECTION

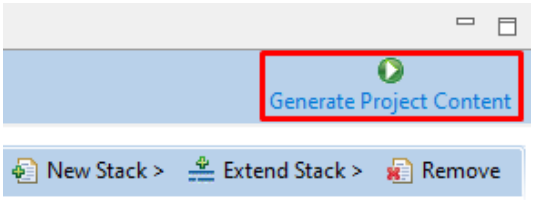
2 Implementing Accelerometer demo

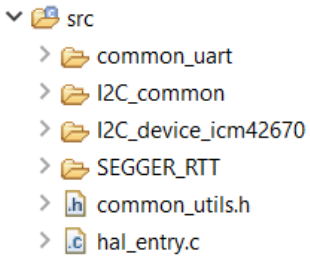
Overview

Following section describes in details steps required to set up an accelerometer demo project for AIK RA8D1 kit.

Procedural Steps

2.1	Create a new project and follow the steps described from 1.1 to 1.12.
2.2	Access the **New Stack** menu and select **Connectivity -> I2C Communication Device (rm_comms_i2c)**. Use **Properties** tab to configure following properties for this new module: • Name g_comms_i2c_device1 • Slave Address 0x68 • Callback rm_icm42670_comms_i2c_callback
2.3	In the I2C Communication Device module, in **g_comms_i2c_bus01 I2C Shared Bus (rm_comms_i2c)** Use **Properties** tab to configure following properties for this new module: • General -> Name g_comms_i2c_bus1 • General -> Channel 1 
2.4	In the I2C Communication Device module, in **Add I2C Communications peripherals** press on the icon -> New and choose **I2C Master (r_iic_master)**. Use **Properties** tab to configure following properties for this new module: • Common -> DTC on Transmission and Reception Enabled • Name g_i2c_master1 • Slave Address 0x68
2.5	In the I2C Communication Device module, in the first **Add DTC Driver for Transmission [optional]** press on the icon -> New and choose **Transfer (r_dtc)**.

2.6	Access the **New Stack** menu again and select **Input -> External IRQ (r_icu)**. Use **Properties** tab to configure following properties for this new module: • Name g_external_irq11_pmod1 • Channel 11 • Trigger Failing • Digital Filtering Enabled • Digital Filtering Sample Clock PCLK/32 • Callback i2c_api_icm42670_irq_callback
2.7	Access the **New Stack** menu again and select **Connectivity -> UART (r_sci_b_uart)**. Use **Properties** tab to configure following properties for this new module: • Common -> DTC Support Enable • General -> Name g_uart3_pmod2 • General -> Channel 3 • Baud -> Max Error (%) 5 • Interrupts -> Callback rm_uart_callback
2.8	In the UART module, in **Add DTC Driver for Transmission [optional]** press on the icon -> New and choose **Transfer (r_dtc)**.
2.9	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**. 
2.10	The FSP Configurator will extract all the necessary drivers and generate the code based on the configuration provided in the **Properties** tab.
2.11	In the **Project Explorer** pane, expand the **src** folder in the project and add the following folders and files than can be found in the demo folder: • I2C_common • SEGGER_RTT • common_uart • I2C_device_icm42670 • common_utils.h • hal_entry.c

2.12	In the **Project Explorer** pane, expand the **src** folder in the project and open **hal_entry.c**. 
2.13	**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).
2.14	At the beginning of hal_entry.c before "void R_BSP_WarmStart(bsp_warm_start_event_t event);" add the #include statement for the following: • `#include "hal_data.h"` • `#include <stdio.h>` • `#include <string.h>` • `#include "common_utils.h"` • `#include "I2C_common/I2C_common.h"` • `#include "I2C_device_icm42670/i2c_api_icm42670.h"` • `#include "common_uart/rm_common_uart.h"` • `#define RM_ICM42670_EXAMPLE_DELAY_50MS 50` • `#define RM_ICM42670_EXAMPLE_DELAY_1US 10` • `#define RM_ICM42670_EXAMPLE_IRQ_ENABLE 1`

2.15	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 operations using the display for the AIK-RA8D1 board:
	
	`#include "hal_data.h"`
	
	`#include <stdio.h>`
	`#include <string.h>`
	`#include "common_utils.h"`
	`#include "I2C_common/I2C_common.h"`
	`#include "I2C_device_icm42670/i2c_api_icm42670.h"`
	`#include "common_uart/rm_common_uart.h"`
	
	`#define RM_ICM42670_EXAMPLE_DELAY_50MS 50`
	`#define RM_ICM42670_EXAMPLE_DELAY_1US 10`
	`#define RM_ICM42670_EXAMPLE_IRQ_ENABLE 1`
	
	`FSP_CPP_HEADER`
	`void R_BSP_WarmStart(bsp_warm_start_event_t event);`
	
	`//void __attribute__((optimize("O0"))) init_i2c_comm(void) ;`
	
	`FSP_CPP_FOOTER`
	
	`#ifdef RTT_DEBUG_ON`
	`char segBuf1[16] ;`
	`char segBuf2[16] ;`
	`#endif`
	
	`/**********...**********//**`
	`* main() is generated by the RA Configuration editor and is used to generate threads if an RTOS is used. This function`
	`* is called by main() when no RTOS is used.`

```
**********************************************************************************************
**********************/
void hal_entry(void)
{
    /* TODO: add your own code here */

#if BSP_TZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif


    fsp_err_t err = FSP_SUCCESS;
    i2c_api_icm42670_raw_data_t   raw_data;
    i2c_api_icm42670_accel_data_t icm42670_accel_data;
    i2c_api_icm42670_gyro_data_t  icm42670_gyro_data;
    i2c_api_icm42670_temp_data_t  icm42670_temp_data;

    #if 0 == RM_ICM42670_EXAMPLE_IRQ_ENABLE
      i2c_api_icm42670_device_status_t device_status;
    #endif

    /* Enable access to the PFS registers. If using r_ioport module then register protection is
automatically
     * handled. This code uses BSP IO functions to show how it is used.
     */
    R_BSP_PinAccessEnable ();
    R_BSP_PinWrite(LED1_BLUE, BSP_IO_LEVEL_HIGH);

    /* Open the uart if it is not already open. */
    err = rm_uart_initialize ();
    if ( err != FSP_SUCCESS)
    {
        R_BSP_PinWrite(LED1_RED, BSP_IO_LEVEL_HIGH);
    }
    else
    {
        R_BSP_PinWrite(LED1_GREEN, BSP_IO_LEVEL_HIGH);
    }
    R_BSP_PinWrite(LED1_BLUE, BSP_IO_LEVEL_LOW);

    /* cursor home */
    printf ("%c[H", 27);

    #ifdef RTT_DEBUG_ON
    // RTT seems not to support cursor home
    //APP_PRINT("\x1B[H");
    #endif

    /* cls  terminal clear screen */
    printf ("%c[2J", 27);
    #ifdef RTT_DEBUG_ON
      APP_PRINT ("RTT_CTRL_CLEAR");
      APP_PRINT (BANNER_INFO);
    #endif

    printf ("UART                 : initialized\r\n");
    #ifdef RTT_DEBUG_ON
      APP_PRINT ("UART                 : initialized\r\n");
    #endif


    /* init the i2c comm interface */
    err = i2c_bus1_comon_init();
    if (err != FSP_SUCCESS)
    {
        R_BSP_PinWrite(LED1_RED, BSP_IO_LEVEL_HIGH);
        __BKPT(0);
    }


    printf ("I2c common interface : initialized\r\n");
    #ifdef RTT_DEBUG_ON
      APP_PRINT ("I2c common interface : initialized\r\n");
```

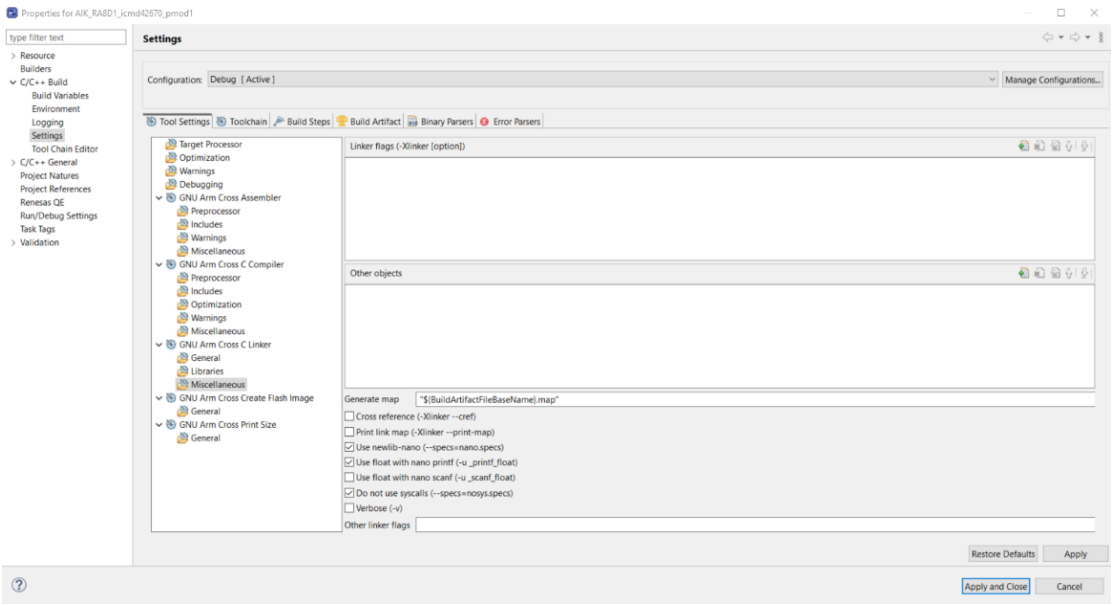
```
    #endif

    /* Open ICM42670  open I2C bus and init sensor */
    err = i2c_api_icm42670_open();

    printf("sensor ack <WhoAmI>  : 0x%2x\r\n",i2c_api_icm42670_get_who_am_i());
    #ifdef RTT_DEBUG_ON
       APP_PRINT ("sensor ack <WhoAmI>  : 0x%2x\r\n",i2c_api_icm42670_get_who_am_i());
    #endif

    if (err != FSP_SUCCESS)
    {
        printf ("I2c sensor setup     : failed\r\n");
        #ifdef RTT_DEBUG_ON
          APP_PRINT ("I2c sensor setup     : failed\r\n");
        #endif
        R_BSP_PinWrite(LED1_RED, BSP_IO_LEVEL_HIGH);
        R_BSP_PinWrite(LED1_GREEN, BSP_IO_LEVEL_LOW);
        R_BSP_PinWrite(LED1_BLUE, BSP_IO_LEVEL_LOW);
        __BKPT(0);
    }

    printf ("I2c ICM42670 setup   : done\r\n");
    #ifdef RTT_DEBUG_ON
      APP_PRINT ("I2c ICM42670 setup   : done\r\n");
    #endif
    /* end */


    {
        i2c_api_icm42670_device_interrupt_cfg_t interrupt_cfg ;
        i2c_api_icm42670_deviceInterruptCfgGet (&interrupt_cfg); //get the recommended settings
;
        //interrupt_cfg.int_config |= 0x01 ; // use active high interrupt
        err =  i2c_api_icm42670_deviceInterruptCfgSet (interrupt_cfg);

        if (err != FSP_SUCCESS)
        {
            R_BSP_PinWrite(LED1_RED, BSP_IO_LEVEL_HIGH);
            __BKPT(0);
        }
        printf ("ICM42670 interrupts  : initialized\r\n");
        #ifdef RTT_DEBUG_ON
          APP_PRINT ("ICM42670 interrupts  : initialized\r\n");
        #endif
    }


   /* Start measurement in data ready mode */
    err = i2c_api_icm42670_measurementStart();
    if (err != FSP_SUCCESS)
    {
        R_BSP_PinWrite(LED1_RED, BSP_IO_LEVEL_HIGH);
        __BKPT(0);
    }
    printf ("ICM42670 measurement : started\r\n");
    #ifdef RTT_DEBUG_ON
      APP_PRINT ("ICM42670 measurement : started\r\n");
    #endif

    /* Open external IRQ */
    err = R_ICU_ExternalIrqOpen(&g_external_irq11_pmod1_ctrl, &g_external_irq11_pmod1_cfg);
    if (err != FSP_SUCCESS)
    {
        R_BSP_PinWrite(LED1_RED, BSP_IO_LEVEL_HIGH);
        __BKPT(0);
    }
    printf ("ICM42670 interrupt   : opened\r\n");
    #ifdef RTT_DEBUG_ON
      APP_PRINT ("ICM42670 interrupt   : opened\r\n");
    #endif

    err = R_ICU_ExternalIrqEnable (&g_external_irq11_pmod1_ctrl);
    if (err != FSP_SUCCESS)
    {
```

```
        R_BSP_PinWrite(LED1_RED, BSP_IO_LEVEL_HIGH);
        __BKPT(0);
    }
    printf ("ICM42670 interrupt   : enabled\r\n");
    #ifdef RTT_DEBUG_ON
      APP_PRINT ("ICM42670 interrupt   : enabled\r\n");
    #endif

    /*
     * Example :
     * Device interrupt : data ready mode
     */

    R_BSP_SoftwareDelay(1500, BSP_DELAY_UNITS_MILLISECONDS);

    //cls  terminal clear screen
    printf ("%c[2J", 27);
    #ifdef RTT_DEBUG_ON
      APP_PRINT (RTT_CTRL_CLEAR);
    #endif

    while (true)
    {
      #if RM_ICM42670_EXAMPLE_IRQ_ENABLE

        /* Wait IRQ callback */
        while (0 == g_i2c_api_irq_flag)
        {
            /* Wait callback */
        }
        g_i2c_api_irq_flag = 0;
     #else
       do
        {
           err = i2c_api_icm42670_deviceStatusGet( &device_status );
           if (err != FSP_SUCCESS)
           {
               R_BSP_PinWrite(LED1_RED, BSP_IO_LEVEL_HIGH);
           }
           else
           {
               R_BSP_PinWrite(LED1_RED, BSP_IO_LEVEL_LOW);
           }
        }
       while (false == device_status.data_ready);
     #endif

#if 1
        /* cursor home */
        printf ("%c[H", 27);
        #ifdef RTT_DEBUG_ON
          // RTT seems not to support cursor home
          //APP_PRINT("\x1B[H");
        #endif

        /* Read Temperature data */
        err = i2c_api_icm42670_tempRead( &raw_data);
        /* Calculate Temperature data
         * measurement is on chip die, lets use an offset of -4.5 deg to get a temperature to
the room temperature.
         *  4.5C == 128* 4 + 64 * 1 ( sensor data will be divided by 128 )
         */
        err = i2c_api_icm42670_tempDataCalculate ( &raw_data, &icm42670_temp_data, -4*128-1*64)
;

        /* Output Temperature data to console */
        printf ("                    \r\n");
        printf ("Temperature: %3.1f [%+3d] degrees Celsius\r\n",
                icm42670_temp_data.temp_data_float,
                icm42670_temp_data.temp_data);

        #ifdef RTT_DEBUG_ON
          snprintf(segBuf1,sizeof(segBuf1)-1,"%3.1f",icm42670_temp_data.temp_data_float);
          snprintf(segBuf2,sizeof(segBuf2)-1,"%+3d",icm42670_temp_data.temp_data);
```

```
          APP_PRINT ("\r\n");
          APP_PRINT ("Temperature: %s [%s] degrees Celsius\r\n", segBuf1, segBuf2);
        #endif

        /* Read Accel data */
        err = i2c_api_icm42670_accelRead (&raw_data);

        /* Calculate Accel data */
        err = i2c_api_icm42670_accelDataCalculate (&raw_data, &icm42670_accel_data);

        /* Output Accel data to console */
        printf ("                    \r\n");
        printf ("Acc_x:  %10.3f\r\n", icm42670_accel_data.accel_x);
        printf ("Acc_y:  %10.3f\r\n", icm42670_accel_data.accel_y);
        printf ("Acc_z:  %10.3f\r\n", icm42670_accel_data.accel_z);

        #ifdef RTT_DEBUG_ON
          APP_PRINT ("\r\n");
          snprintf(segBuf1,sizeof(segBuf1)-1,"%10.3f",icm42670_accel_data.accel_x);
          APP_PRINT ("Acc_x:  %s\r\n", segBuf1);
          snprintf(segBuf1,sizeof(segBuf1)-1,"%10.3f",icm42670_accel_data.accel_y);
          APP_PRINT ("Acc_y:  %s\r\n", segBuf1);
          snprintf(segBuf1,sizeof(segBuf1)-1,"%10.3f",icm42670_accel_data.accel_z) ;
          APP_PRINT ("Acc_z:  %s\r\n", segBuf1);
        #endif

        /* Read Gyro data */
        err = i2c_api_icm42670_gyroRead (&raw_data);

        /* Calculate Gyro data */
        err = i2c_api_icm42670_gyroDataCalculate (&raw_data, &icm42670_gyro_data);

        /* Output Gyro data to console */
        printf ("                   \r\n");
        printf ("Gyro_x: %10.3f\r\n", icm42670_gyro_data.gyro_x);
        printf ("Gyro_y: %10.3f\r\n", icm42670_gyro_data.gyro_y);
        printf ("Gyro_z: %10.3f\r\n", icm42670_gyro_data.gyro_z);

        #ifdef RTT_DEBUG_ON
          APP_PRINT ("\r\n");
          snprintf(segBuf1,sizeof(segBuf1)-1,"%10.3f",icm42670_gyro_data.gyro_x);
          APP_PRINT ("Gyro_x: %s\r\n", segBuf1);
          snprintf(segBuf1,sizeof(segBuf1)-1,"%10.3f",icm42670_gyro_data.gyro_y);
          APP_PRINT ("Gyro_y: %s\r\n", segBuf1);
          snprintf(segBuf1,sizeof(segBuf1)-1,"%10.3f",icm42670_gyro_data.gyro_z);
          APP_PRINT ("Gyro_z: %s\r\n", segBuf1);
        #endif
        {
            static uint16_t mode = BSP_IO_LEVEL_HIGH ;
            mode = mode == BSP_IO_LEVEL_HIGH ?  BSP_IO_LEVEL_LOW : BSP_IO_LEVEL_HIGH ;
            R_BSP_PinWrite(LED1_GREEN, mode);
        }
#endif
    }

}

/*******************************************************************************************************************//**
 * This function is called at various points during the startup process.  This implementation uses the event that is
 * called right before main() to set up the pins.
 *
 * @param[in]  event    Where at in the start up process the code is currently at
 **********************************************************************************************************************/
void R_BSP_WarmStart(bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_RESET == event)
    {
#if BSP_FEATURE_FLASH_LP_VERSION != 0

        /* Enable reading from data flash. */
        R_FACI_LP->DFLCTL = 1U;
```

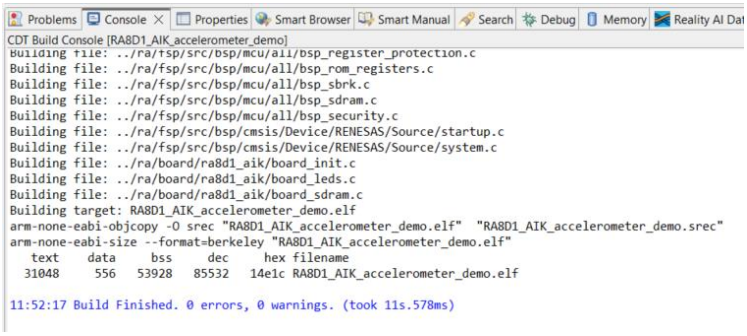
```
        /* Would normally have to wait tDSTOP(6us) for data flash recovery. Placing the enable
here, before clock and
         * C runtime initialization, should negate the need for a delay since the
initialization will typically take more than 6us. */
#endif
    }

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

        /* Configure pins. */
        R_IOPORT_Open (&g_ioport_ctrl, &IOPORT_CFG_NAME);
    }
}

#if BSP_TZ_SECURE_BUILD

FSP_CPP_HEADER
BSP_CMSE_NONSECURE_ENTRY void template_nonsecure_callable ();

/* Trustzone Secure Projects require at least one nonsecure callable function in order to build
(Remove this if it is not required to build). */
BSP_CMSE_NONSECURE_ENTRY void template_nonsecure_callable ()
{

}
FSP_CPP_FOOTER

#endif
```

2.16 Right-click the project in the **Project Explorer** and select **Properties** form the context menu, then navigate to **C/C++ Build -> Settings**. Make sure you're on the tool **Setting -> GNU Arm Cross Linker -> Miscellaneous** tab and click on the **Use float with nano printf (-u_printf_float)** also enable **Do not use syscalls.**



2.17 The project is now ready to compile. Press the "hammer" icon to start building the project.

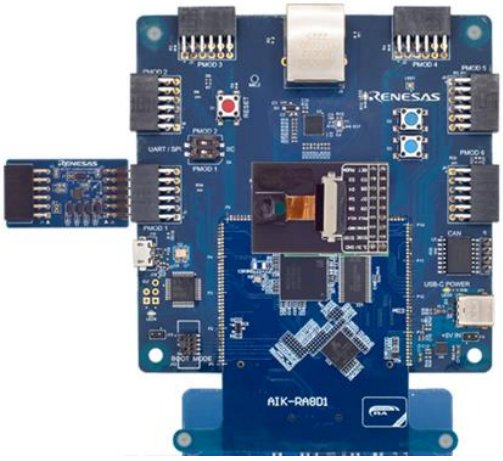
2.18 Once the build has finished, the **Console** pane in the lower-right corner of e² studio will report zero error and warnings:



2.19 Connect PMOD2 Pin2 & Pin 3 with the USB2Serial TX & RX pins of the dongle respectively to enable UART output through Teraterm.

Pin	Signal/Bus SPI	Description UART
2	P707	TXD
3	P706	RXD

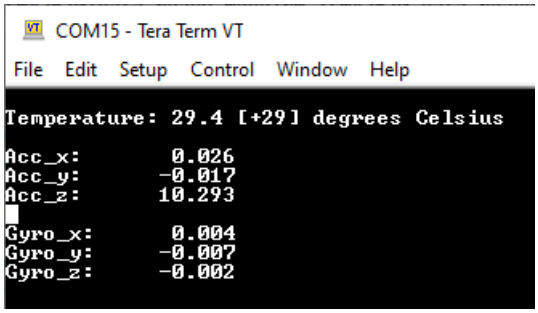
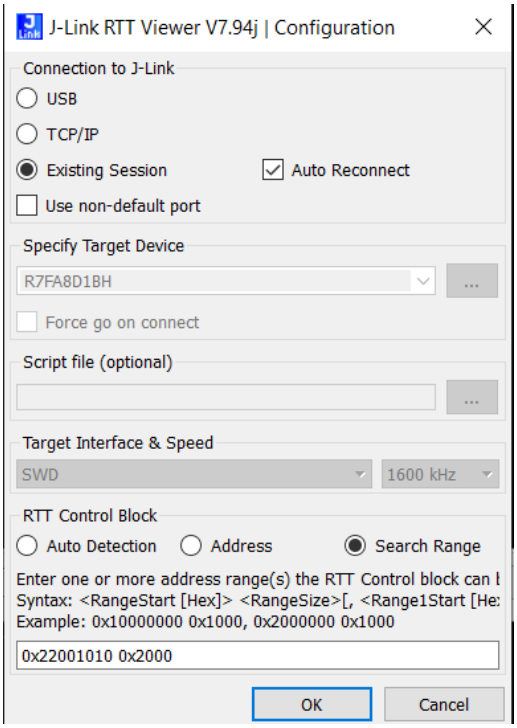
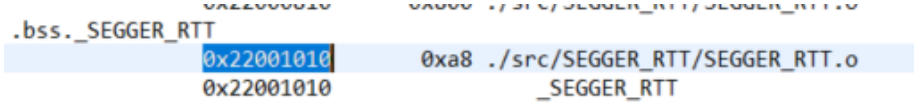
2.20 Check that the Accelerometer is connected to PMOD1 as seen below.



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

2.22 Bring up the serial terminal window to observe the debug output from the AIK kit.

2.23 Click the **Resume** button or press **F8** on the keyboard to start the application. Press **Resume** or **F8** again to resume the application and begin executing user code.

2.24	Go back to the serial terminal window which should now be populated with debug output from the AIK kit. First line shows the temperature in degrees Celsius , the rest lines show the data of the accelerometers x,y,z axis and the data from the gyroscope x,y,z axis. 
2.25	To view the log from J-Link RTT Viewer, use: Connection to J-Link → Existing Session RTT Control Block → Search Range and in the Range add: 0x22001010 0x2000  Else go to e2studio RA8D1_AIK_accelerometer_demo.map, find 0xa8 ./src/SEGGER_RTT/SEGGER_RTT.o and get the value next to it and in the J_Link RTT Viewer, use: Connection to J-Link → Existing Session RTT Control Block → Address and use the value you found above. 

2.26	Click the **Terminate** button or press **Ctrl + F2** on the keyboard to stop the application and terminate the debug session. 
2.27	Full project is also available in GitHub under https://github.com/renesas/aiot-ready/tree/master/kits/RA8D1_aik/qse

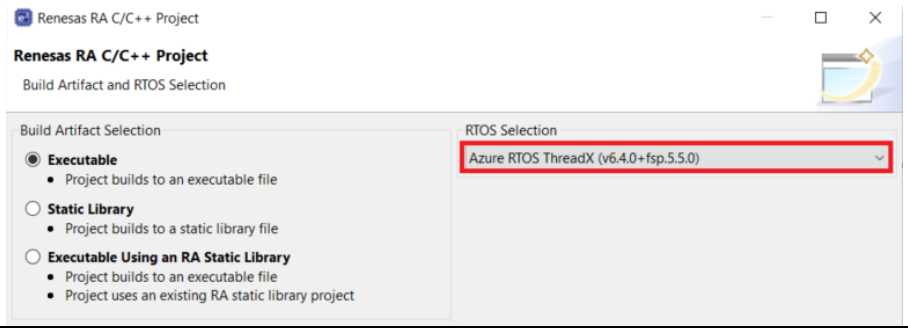
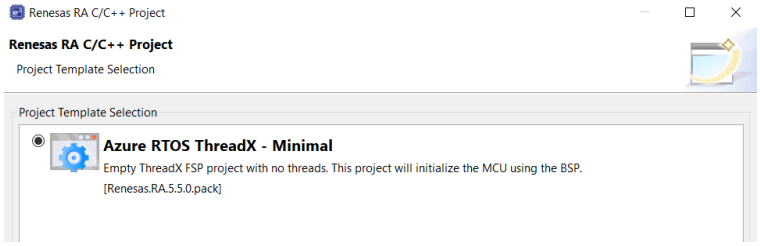
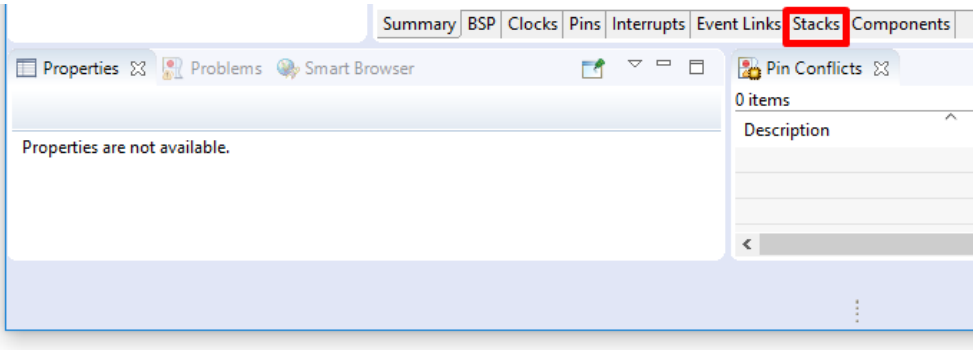
END OF SECTION

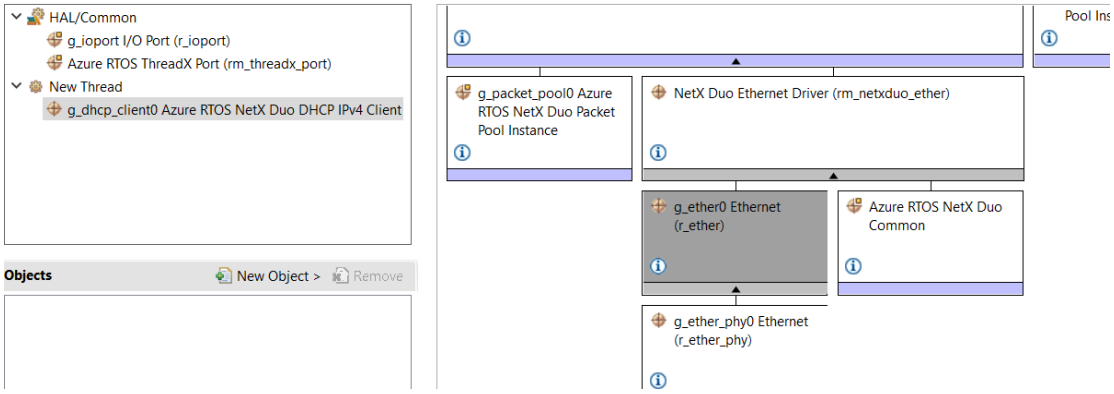
3 Implementing Ethernet demo

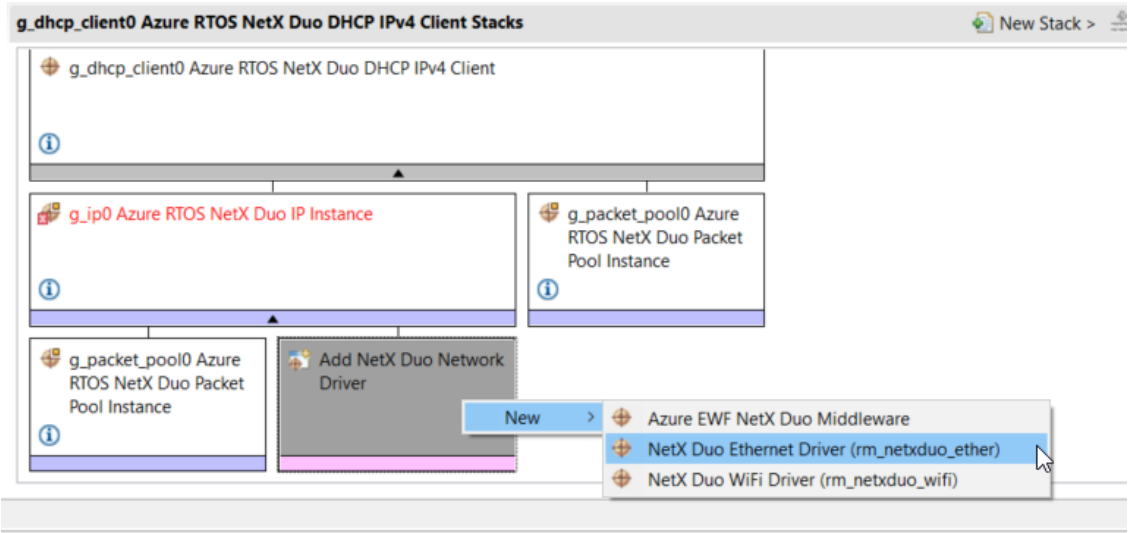
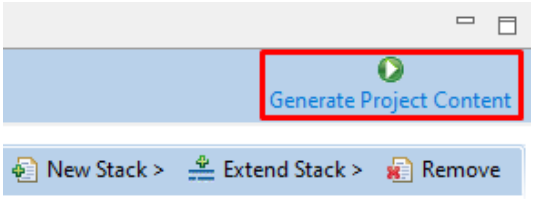
Overview

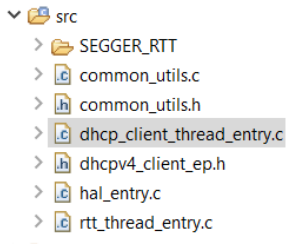
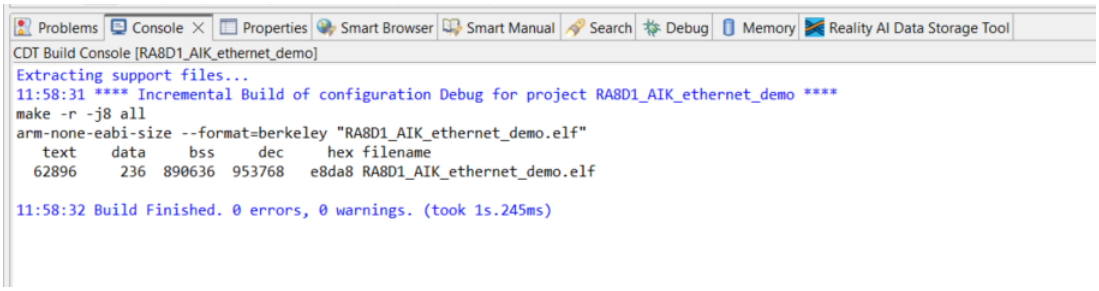
Following section describes in details steps required to set up an Ethernet demo project for AIK RA8D1 kit.

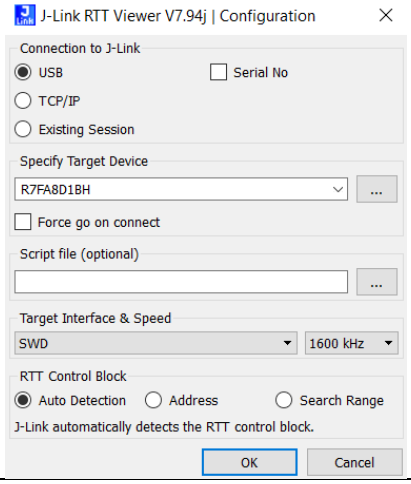
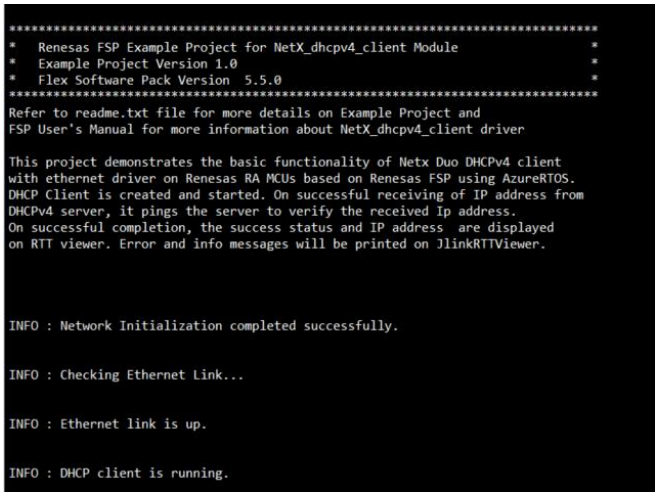
Procedural Steps

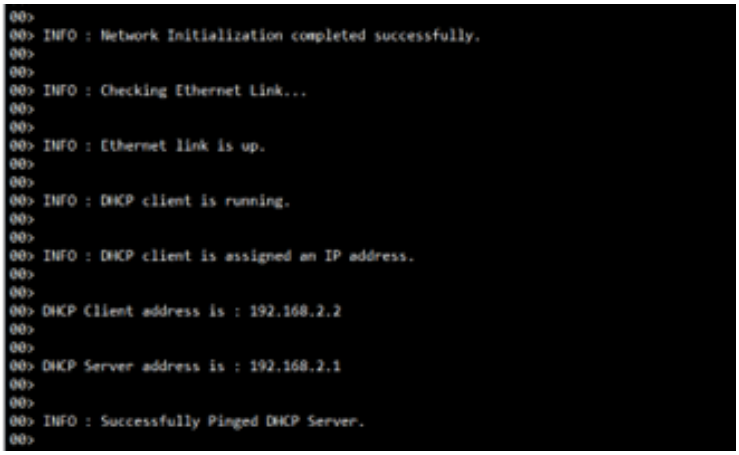
3.1	Prerequisites for this demo are: • Router with ethernet connection • Ethernet Cable
3.2	Create a new project and follow the steps described from 1.1 to 1.12.
3.3	On the next window, leave **Executable** and **Azure RTOS ThreadX** selected. Click **Next**.
3.4	When prompted to open the **FSP Configuration perspective**, click **Open Perspective**. The project is now set up to begin evaluation and development using the AIK kit. 
3.5	On the final page of the new project wizard select **Azure RTOS ThreadX – Minimal** and click **Finish**. 
3.6	Once the 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. 

3.7	Access the **New Thread** menu and select **New Thread**. Use **Properties** tab to configure following properties for this new module: • Thread -> Symbol rtt_thread • Thread -> Name RTT Thread • Thread -> Stack size (byte) 1024 • Thread -> Priority 4
3.8	Access the **New Thread** menu and select **New Thread**. Use **Properties** tab to configure following properties for this new module: • Thread -> Symbol dhcp_client_thread • Thread -> Name DHCP Client Thread • Thread -> Stack size (byte) 2048 • Thread -> Priority 3
3.9	Access the **New Stack** menu and select **Networking -> Azure RTOS NetX Duo DHCP IPv4 Client**. Use **Properties** tab to configure following properties for this new module: • DHCP -> Client -> IPv4 -> Persistent client state Enable • FTP -> Server -> Binary left shift as multiplier for next retry duration 2
3.10	In the Azure RTOS NetX Duo DHCP IPv4 Client module, press **Add NetX Duo Network Driver -> New -> NetX Duo Ethernet Driver (rm_netxduo_ether)**. 
3.11	In the Azure RTOS NetX Duo DHCP IPv4 Client module, press **Add NetX Duo Packet Pool -> Use -> g_packet_pool0 Azure RTOS NetX Duo Packet Pool Instance**. Use **Properties** tab to configure following properties for this new module: • DHCP -> Client -> IPv4 -> Persistent client state Enable

3.12	In the Azure RTOS NetX Duo DHCP Duo IP Instance e, press **Add NetX Duo Network Driver -> New -> NetX Duo Ethernet Driver (rm_netduo_ether)** 
3.13	In the **g_ether_phy0 Ethernet (r_ether_phy)** module. Use **Properties** tab to configure following properties for this new module: • Module- g_ether_phy0 Ethernet (r_ether_phy) ->PHY-LSI Address 7 • Module- Phy LSI type ICS 1894
3.14	Go to Pins and Select Pin Configuration: **RA8D1-AIK_ETH.pincfg**
3.15	Go to Pins **Connectivity:ETHER_RMII→ETHER_RMII** and in **Operation mode** value add **RMII.**
3.16	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**. 
3.17	The FSP Configurator will extract all the necessary drivers and generate the code based on the configuration provided in the **Properties** tab.
3.18	In the **Project Explorer** pane, expand the **src** folder in the project, remove rtt_thread_entry.c file and add the following folders and files that can be found in the demo folder: • SEGGER_RTT • common_utils.h • common_utils.c • dhcp_client_thread_entry.c • dhcpv4_client_ep.h • rtt_Thread_entry.c

	• hal_entry.c
3.19	In the **Project Explorer** pane, expand the **src** folder in the project and open **hal_entry.c**. 
3.20	The project is now ready to compile. Press the “hammer” icon to start building the project. 
3.21	Once the build has finished, the **Console** pane in the lower-right corner of e^2 studio will report zero error and warnings: 
3.22	The application is now ready to be programmed and run on the AIK kit. Press the “bug” icon to begin the debug session. 

3.23	Bring up the J-Link RTT Viewer window to observe the debug output from the AIK kit. Go to File -> Connect and make the following changes: • Connection to J-Link USB • Specify Target Device R7FA8D1BH • Target interface & Speed SWD 4000 kHz 
3.24	Click the **Resume** button or press **F8** on the keyboard to start the application. Press **Resume** or **F8** again to resume the application and begin executing user code. 
3.25	Go back to the terminal window which should now be populated with debug output from the AIK kit. 

3.26	Insert the Ethernet cable to the AIK kit and the terminal prints the following information: • Network Initialization completed successfully. • Checking Ethernet Link... • Ethernet link is up. • DHCP client is running. • DHCP client is assigned an IP address • DHCP Client address is : 192.168.2.2 • DHCP Server address is : 192.168.2.1 • INFO : Successfully Pinged DHCP Server. **Note**: Values in DHCP Client/Server address may vary. 
3.27	Click the **Terminate** button or press **Ctrl + F2** on the keyboard to stop the application and terminate the debug session. 
3.28	Full project is also available in GitHub under https://github.com/renesas/aiot-ready/tree/master/kits/RA8D1_aik/qse

For further information and inquiries please contact: rai-cs@dm.renesas.com

END OF SECTION