RA Family, RX Family, RL78 Family, RZ Family

Sensor Software Combination Manual

Introduction

This application note describes code changes required to use the multiple sensor software combinations and runs on certain MCUs of the RA family, RX family, RL78 family and RZ family.

Target Devices

RA6M4 Group

RX65N Group

RL78/G23 Group

RL78/G14 Group

RZ/G2L Group

Reference Documents

HS300x Sample Software Manual (R01AN5897)

HS400x Sample Software Manual (R01AN6333)

FS2012 Sample Software Manual (R01AN6047)

FS3000 Sample Software Manual (R01AN5898)

FS1015 Sample Software Manual (R01AN6049)

ZMOD4xxx Sample Software Manual (R01AN5899)

OB1203 Sample Software Manual (R01AN6311)

Trademarks

FreeRTOS™ and FreeRTOS.org™ are trade marks of Amazon Web Services, Inc.

Microsoft® Azure RTOS is trade marks of the Microsoft group of companies. All other trade marks are property of their respective owners.

Contents

1. Overview

To use the combination of multiple sensor software sample projects, the following changes are required.

- I2C bus configuration changes
- File configuration and code changes according to RTOS

2. I2C Shared Bus

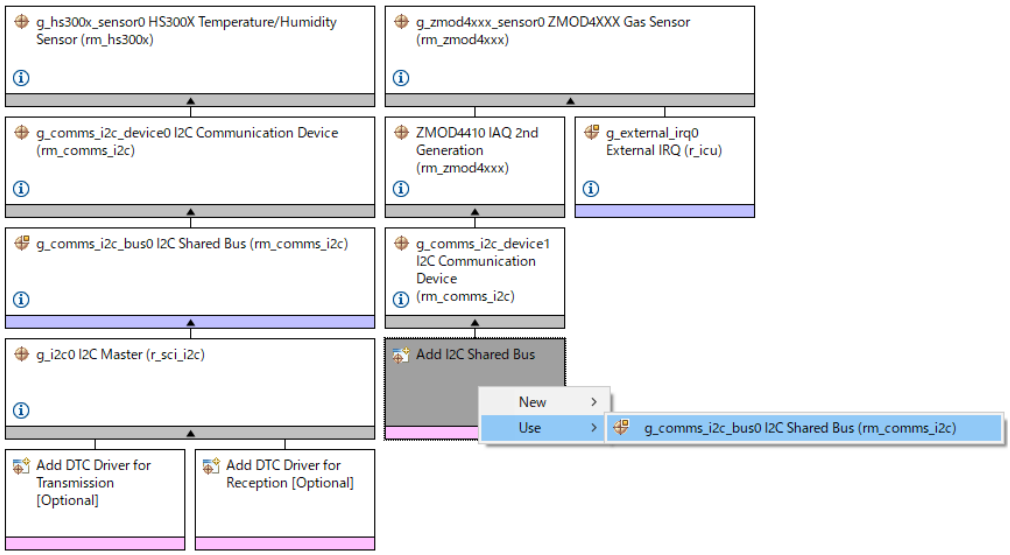
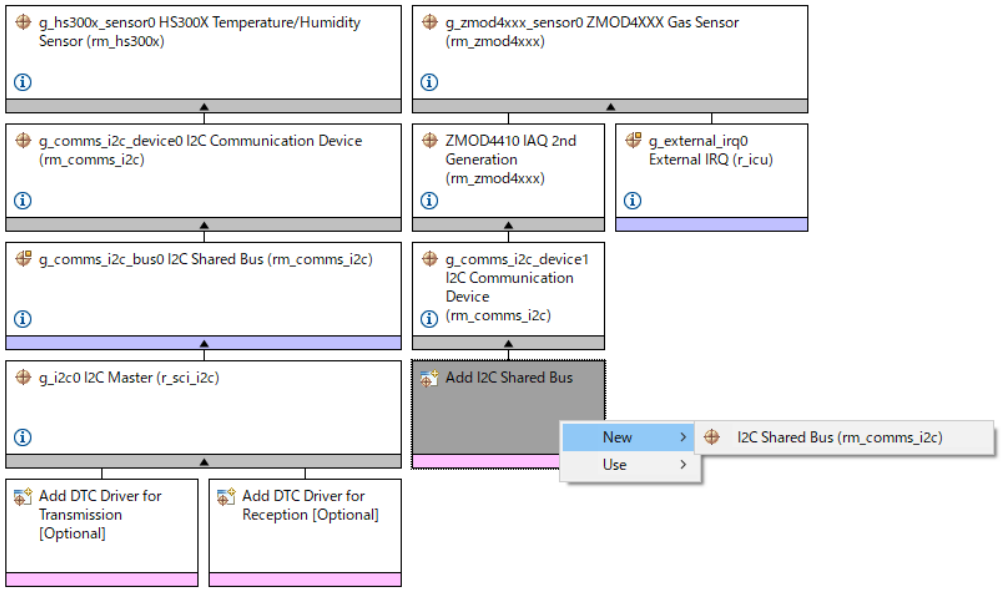
This chapter describes the processes required when multiple sensors share the same I2C bus and when different I2C buses are used. Configuration settings differ depending on the MCU used.

*The explanation is based on the example of using HS3001 in combination with ZMOD4410. The same method is used for the combination of other sensors.

2.1 RA, RZ

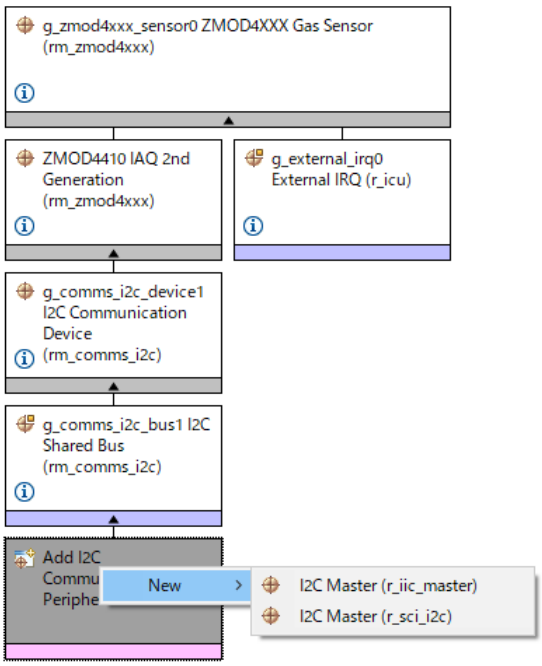
With HS300X stack is added, ZMOD4xxx stack is added. “Add I2C Shared Bus” will be available as shown below.

- Select “Use” to share the same I2C bus between HS3001 and ZMOD4410
- Select “New” to use different I2C buses for HS3001 and ZMOD4410

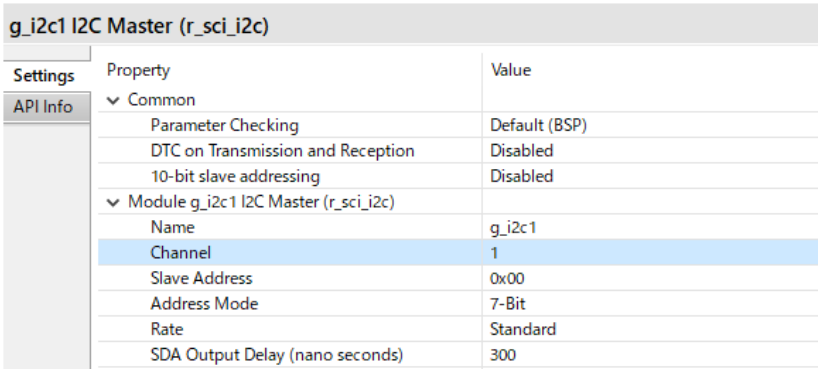
The same I2C bus is shared	
The different I2C bus is used	

When you select "New" in "Add I2C Shared Bus", "Add I2C Communications Peripheral" will be available as shown below.

If the different I2C buses are used, select the peripheral function to be used.

The different I2C bus is used	

Set the channel to be used in the properties of the selected peripheral function.

The different I2C bus is used	

2.2 RL78/G14

Before applying the changes in this chapter, perform 3.1.1 Copy and paste files.

Please open r_(sensor_name)_rl_config.h file and set
the “RM_HS300X_CFG_DEVICE0_COMMS_INSTANCE” to g_comms_i2c_device0,
the ”RM_ZMOD4XXX_CFG_DEVICE0_COMMS_INSTANCE” to g_comms_i2c_device1.(to use HS3001 and ZMOD4410 sensors)

```
/* SPECIFY USING COMMUNICATION LINE INSTANCE FOP DEVICE0 */
#define RM_ZMOD4XXX_CFG_DEVICE0_COMMS_INSTANCE            (g_comms_i2c_device1)
```

Please open r_comms_i2c_rl_config.h file and set the “COMMS_I2C_CFG_DEVICE_NUM_MAX” to 2 (to use HS3001 and ZMOD4410 sensors)

- If the I2C bus is shared, set the “COMMS_I2C_CFG_BUS_NUM_MAX” to 1
- If the different I2C buses are used, set the “COMMS_I2C_CFG_BUS_NUM_MAX” to 2

The same I2C bus is shared	`/* SPECIFY NUMBER OF BUSES */`
	`#define COMMS_I2C_CFG_BUS_NUM_MAX (1)`
	
	`/* SPECIFY NUMBER OF DEVICES */`
	`#define COMMS_I2C_CFG_DEVICE_NUM_MAX (2)`

The different I2C bus is used	`/* SPECIFY NUMBER OF BUSES */`
	`#define COMMS_I2C_CFG_BUS_NUM_MAX (2)`
	
	`/* SPECIFY NUMBER OF DEVICES */`
	`#define COMMS_I2C_CFG_DEVICE_NUM_MAX (2)`

Next, set the “COMMS_I2C_CFG_DEVICE1_BUS_CH”

- If the I2C bus is shared, set “g_comms_i2c_bus0_extended_cfg” which is the same bus number
- If the different I2C buses are used, set “g_comms_i2c_bus1_extended_cfg” which is the different bus number

Also, set the” COMMS_I2C_CFG_DEVICE1_SLAVE_ADDR” to 0x32,
the “COMMS_I2C_CFG_DEVICE1_CALLBACK” to rm_zmod4xxx_callback0.

The same I2C bus is shared

```
/* For Device No.0 */
#define COMMS_I2C_CFG_DEVICE0_BUS_CH            (g_comms_i2c_bus0_extended_cfg)
#define COMMS_I2C_CFG_DEVICE0_SLAVE_ADDR        (0x44) /* Slave address */
#define COMMS_I2C_CFG_DEVICE0_CALLBACK          (rm_hs300x_callback0) /* Callback function */

/* For Device No.1 */
#define COMMS_I2C_CFG_DEVICE1_BUS_CH            (g_comms_i2c_bus0_extended_cfg)
#define COMMS_I2C_CFG_DEVICE1_SLAVE_ADDR        (0x32) /* Slave address */
#define COMMS_I2C_CFG_DEVICE1_CALLBACK          (rm_zmod4xxx_callback0) /* Callback function */
```

The different I2C bus is used

```
/* For Device No.0 */
#define COMMS_I2C_CFG_DEVICE0_BUS_CH            (g_comms_i2c_bus0_extended_cfg)
#define COMMS_I2C_CFG_DEVICE0_SLAVE_ADDR        (0x44) /* Slave address */
#define COMMS_I2C_CFG_DEVICE0_CALLBACK          (rm_hs300x_callback0) /* Callback function */

/* For Device No.1 */
#define COMMS_I2C_CFG_DEVICE1_BUS_CH            (g_comms_i2c_bus1_extended_cfg)
#define COMMS_I2C_CFG_DEVICE1_SLAVE_ADDR        (0x32) /* Slave address */
#define COMMS_I2C_CFG_DEVICE1_CALLBACK          (rm_zmod4xxx_callback0) /* Callback function */
```

Next, set the “COMMS_I2C_CFG_BUS1_DRIVER_TYPE”, and "COMMS_I2C_CFG_BUS1_DRIVER_CH“.

- If the I2C bus is shared, there is no change
- If the different I2C buses are used, set the Driver Type and channel number to be used

The different I2C bus is used

```
/* For Bus No.0 */
#define COMMS_I2C_CFG_BUS0_DRIVER_TYPE          (COMMS_DRIVER_I2C) /* Driver type of I2C Bus */
#define COMMS_I2C_CFG_BUS0_DRIVER_CH            (0) /* Channel No. */

/* For Bus No.1 */
#define COMMS_I2C_CFG_BUS1_DRIVER_TYPE          (COMMS_DRIVER_I2C) /* Driver type of I2C Bus */
#define COMMS_I2C_CFG_BUS1_DRIVER_CH            (1) /* Channel No. */
```

If the different I2C buses are used, the callback function must be called.

Open the r_cg_serial_user.c and add the call of the rm_comms_i2c_bus1_callback() to the callback function of the channel to be used.

Specify the "false" parameter for the transmission and reception end callback functions and the "true" parameter for the error callback function.

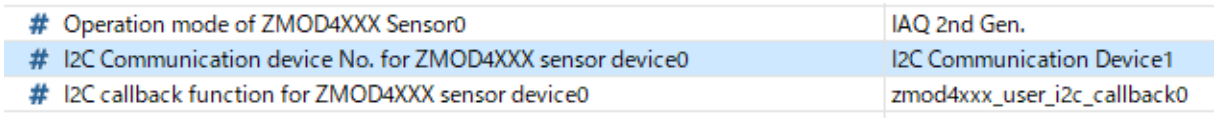
```
/***********************************************************************************************************
* Function Name: r_iic00_callback_master_error
* Description  : This function is a callback function when IIC00 master error occurs.
* Arguments    : flag -
*                    status flag
* Return Value : None
***********************************************************************************************************/
static void r_iic00_callback_master_error(MD_STATUS flag)
{
    /* Start user code. Do not edit comment generated here */
    rm_comms_i2c_bus1_callback(true);
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************************************
* Function Name: r_iic00_callback_master_receiveend
* Description  : This function is a callback function when IIC00 finishes master reception.
* Arguments    : None
* Return Value : None
***********************************************************************************************************/
static void r_iic00_callback_master_receiveend(void)
{
    /* Start user code. Do not edit comment generated here */
    rm_comms_i2c_bus1_callback(false);
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************************************
* Function Name: r_iic00_callback_master_sendend
* Description  : This function is a callback function when IIC00 finishes master transmission.
* Arguments    : None
* Return Value : None
***********************************************************************************************************/
static void r_iic00_callback_master_sendend(void)
{
    /* Start user code. Do not edit comment generated here */
    rm_comms_i2c_bus1_callback(false);
    /* End user code. Do not edit comment generated here */
}
```

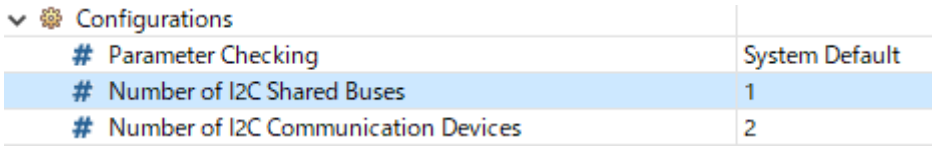
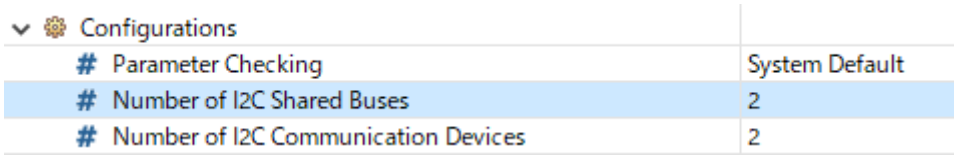
2.3 RX, RL78/G2x

Please select r_(sensor_name) module and set the "I2C communication device No. for HS300x sensor device0" to I2C Communication Device0, the "I2C communication device No. for ZMOD4XXX sensor device0" to I2C Communication Device1. (to use HS3001 and ZMOD4410 sensors)



Please select r_comms_i2c module and set the “Number of I2C Communication Devices” to 2 (to use HS3001 and ZMOD4410 sensors)

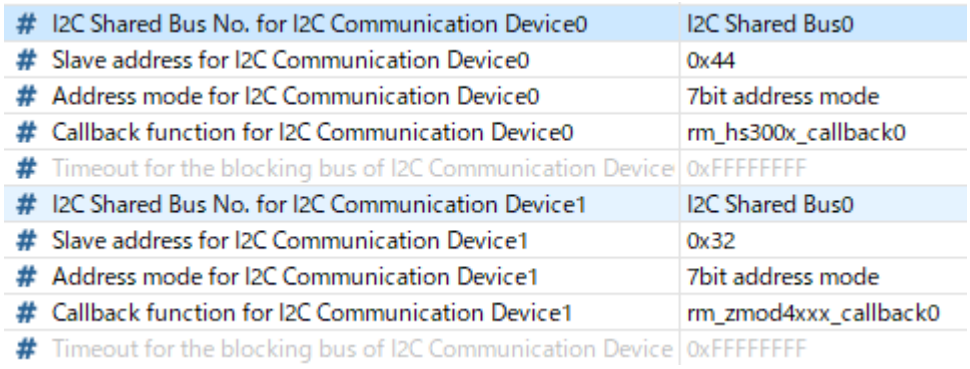
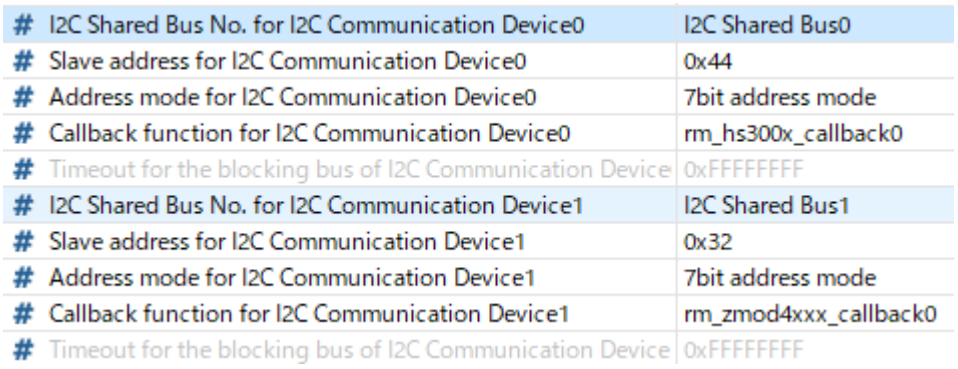
- If the I2C bus is shared, set the “Number of I2C Shared Buses” to 1
- If the different I2C buses are used, set the “Number of I2C Shared Buses” to 2

The same I2C bus is shared	
The different I2C bus is used	

Next, set the “I2C Shared Bus No. for I2C Communication Device1”.

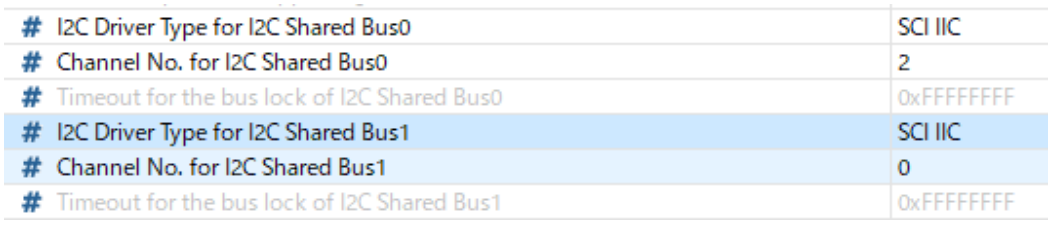
- If the I2C bus is shared, set “I2C Shared Bus0” which is the same I2C Shared Bus number
- If the different I2C buses are used, set “I2C Shared Bus1” which is the different I2C Shared Bus number

Also, set the ”Slave address for I2C Communication Device1” to 0x32, the “Callback function for I2C Communication Device1” to rm_zmod4xxx_callback0.

The same I2C bus is shared	
The different I2C bus is used	

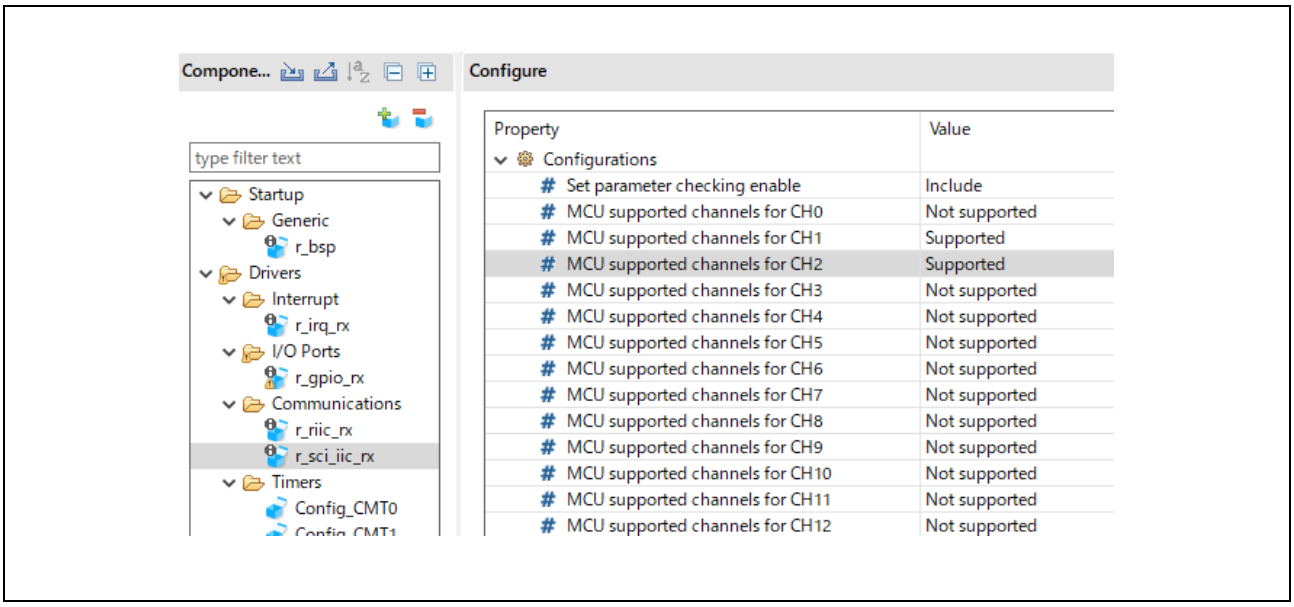
Next, set the ”I2C Driver Type for I2C Shared Bus1”, and ”Channel No. for I2C Shared Bus1”.

- If the I2C bus is shared, there is no change.
- If the different I2C buses are used, set the Driver Type and channel number to be used.

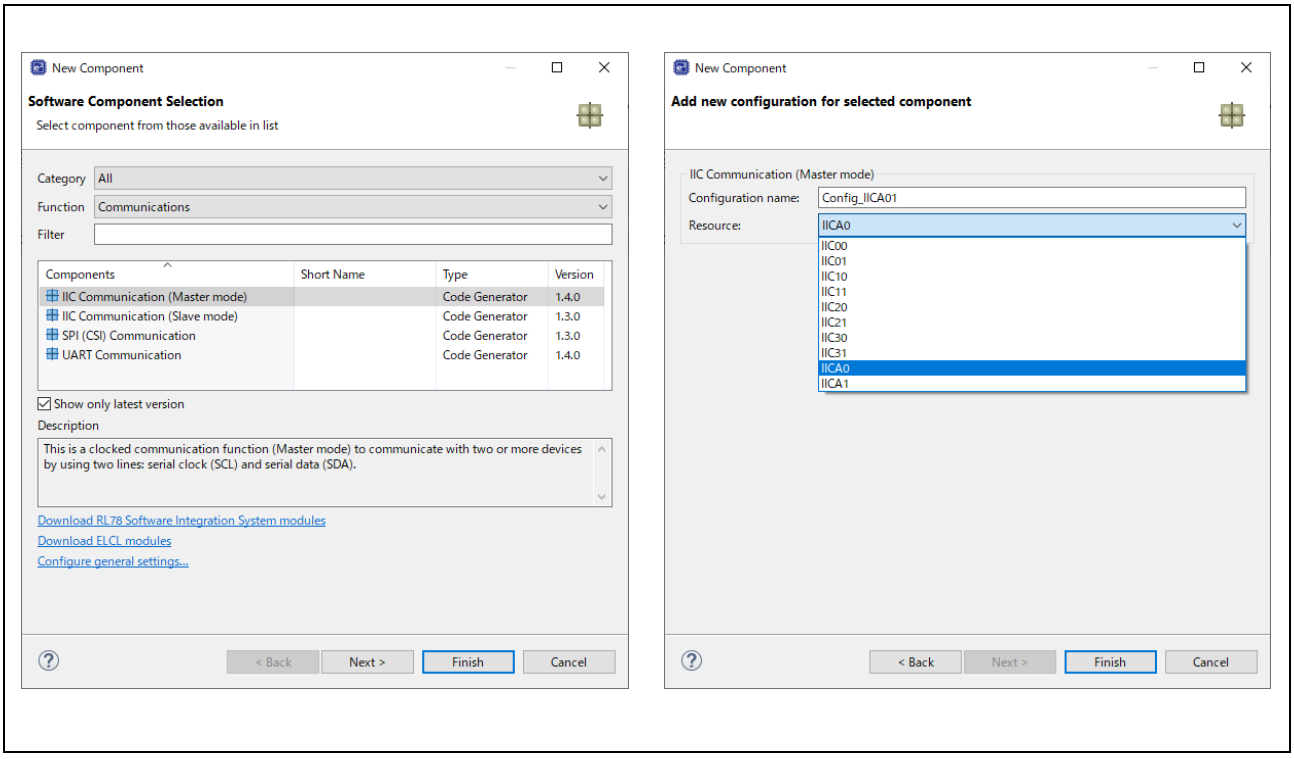
The different I2C bus is used	

If the different I2C buses are used, add components and change settings.

When using the RX project, select r_sci_iic_rx or r_riic_rx and enable the channel according to the Driver Type and channel number that set to I2C Shared Bus1.



When using the RL78/G2x project, select the component according to the Driver Type and channel number set for I2C Shared Bus1 by “add component”.



2.4 I2C bus setup (RA, RX, RZ Only)

If the different I2C buses are used, it is necessary to initialize the I2C driver.

Note: The code example in the text box from the sensor software sample projects for RA. The same method is used for the combination of other sensors.

When using the RL78 MCU, there is no need to initialize the I2C driver, because Code Generator and Smart Configurator generate initialization processing for the I2C driver.

2.4.1 NonOS

Copy and paste g_comms_i2c_bus0_quick_setup() in the C file where main() or hal_entry() is declared.

Change the name of this function to g_comms_i2c_bus1_quick_setup() and the I2C instance referenced in this function to g_comms_i2c_bus1_extended_cfg.

```
/* Quick setup for g_comms_i2c_bus1. */
void g_comms_i2c_bus1_quick_setup(void)
{
    fsp_err_t err;
    i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
                            g_comms_i2c_bus1_extended_cfg.p_driver_instance;

    /* Open I2C driver, this must be done before calling any COMMS API */
    err = p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
          p_driver_instance->p_cfg);
    if (FSP_SUCCESS != err)
    {
        demo_err();
    }
}
```

Call g_comms_i2c_bus1_quick_setup() in main() or hal_entry().

2.4.2 FreeRTOS/Azure

Note: When using the RX FreeRTOS project, refer to 2.4.1 NonOS.

Examples of function definitions are described separately for FreeRTOS and Azure.

Copy and paste g_comms_i2c_bus0_quick_setup() in sensor_thread_common.c.

Change the name of this function to g_comms_i2c_bus1_quick_setup() and the I2C instance referenced in this function to g_comms_i2c_bus1_extended_cfg.

Also, change the setup flag that set in the function to g_comms_i2c_bus1_setup.

- FreeRTOS

```
/* Quick setup for g_comms_i2c_bus1. */
void g_comms_i2c_bus1_quick_setup(TaskHandle_t task)
{
    fsp_err_t err;
    i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
g_comms_i2c_bus1_extended_cfg.p_driver_instance;

    /* Open I2C driver, this must be done before calling any COMMS API */
    err = p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
                                         p_driver_instance->p_cfg);
    if (FSP_SUCCESS != err)
    {
        vTaskDelete(task);
    }

    /* Create a semaphore for blocking if a semaphore is not NULL */
    if (NULL != g_comms_i2c_bus1_extended_cfg.p_blocking_semaphore)
    {
        *(g_comms_i2c_bus1_extended_cfg.p_blocking_semaphore->p_semaphore_handle)
            = xSemaphoreCreateCountingStatic((UBaseType_t) 1, (UBaseType_t) 0,
       g_comms_i2c_bus1_extended_cfg.p_blocking_semaphore->p_semaphore_memory);
    }

    /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
    if (NULL != g_comms_i2c_bus1_extended_cfg.p_bus_recursive_mutex)
    {
        *(g_comms_i2c_bus1_extended_cfg.p_bus_recursive_mutex->p_mutex_handle)
            = xSemaphoreCreateRecursiveMutexStatic
              (g_comms_i2c_bus1_extended_cfg.p_bus_recursive_mutex->p_mutex_memory);
    }

    /* Set setup flag */
    g_comms_i2c_bus1_setup = true;
}
```

- Azure

```
/* Quick setup for g_comms_i2c_bus1. */
void g_comms_i2c_bus1_quick_setup(TX_THREAD* thread_ptr)
{
    fsp_err_t err;
    i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
                            g_comms_i2c_bus1_extended_cfg.p_driver_instance;

    /* Open I2C driver, this must be done before calling any COMMS API */
    err = p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
          p_driver_instance->p_cfg);
    if (FSP_SUCCESS != err)
    {
        tx_thread_delete(thread_ptr);
    }

    /* Create a semaphore for blocking if a semaphore is not NULL */
    if (NULL != g_comms_i2c_bus1_extended_cfg.p_blocking_semaphore)
    {
       tx_semaphore_create(g_comms_i2c_bus1_extended_cfg.p_blocking_semaphore->
       p_semaphore_handle,
       g_comms_i2c_bus1_extended_cfg.p_blocking_semaphore->p_semaphore_name,
        (ULONG) 0);
    }

    /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
    if (NULL != g_comms_i2c_bus1_extended_cfg.p_bus_recursive_mutex)
    {
       tx_mutex_create(g_comms_i2c_bus1_extended_cfg.p_bus_recursive_mutex->
       p_mutex_handle,
       g_comms_i2c_bus1_extended_cfg.p_bus_recursive_mutex->p_mutex_name,
       TX_INHERIT);
    }

    /* Set setup flag */
    g_comms_i2c_bus1_setup = true;
}
```

Next, define the setup flag.

```
bool g_comms_i2c_bus1_setup = false;
```

Add the prototype declaration and the extern declaration in sensor_thread_common.h as follows.

```
void g_comms_i2c_bus1_quick_setup(TX_THREAD* thread_ptr);
extern bool g_comms_i2c_bus1_setup;
```

In the (sensor_name)_sensor_thread_entry() in the sensor task or thread that uses I2C Shared Bus1, change the calling process of g_comms_i2c_bus0_quick_setup() as follow.

```
if(!g_comms_i2c_bus1_setup)
{
    /* Open the Bus */
    g_comms_i2c_bus1_quick_setup(&zmod4410_sensor_thread);
}
```

3. Code change procedure

This chapter describes how to change the code to operate while switching sensors when I2C communication is completed and when waiting for an IRQ signal.

Before changing codes, it is necessary to configure the settings on Smart Configurator or Code Generator with reference to each sample project. (In case of RL78, RX MCU, functions such as callbacks need to be added to the generated driver code.)

*The explanation is based on an example using RA as the MCU and HS3001 and ZMOD4410 as the sensors. If different changes are required depending on the type of MCU or sensor, explanations are added.

When using BSPv1.30 or earlier on RL78/G2x, the following changes are required to prevent multiple definitions.

Change the variable bsp_delay_time defined in rm_(sensor name)_common.c in each sensor module as follows. After the change, delete the rm_(sensor name)_common_(complier name).asm in each sensor module.

Before	`const unsigned long long bsp_delay_time[] = {`
	`  1,`
	`  1000,`
	`  1000000`
	`};`

After	`extern const unsigned long long bsp_delay_time[];`

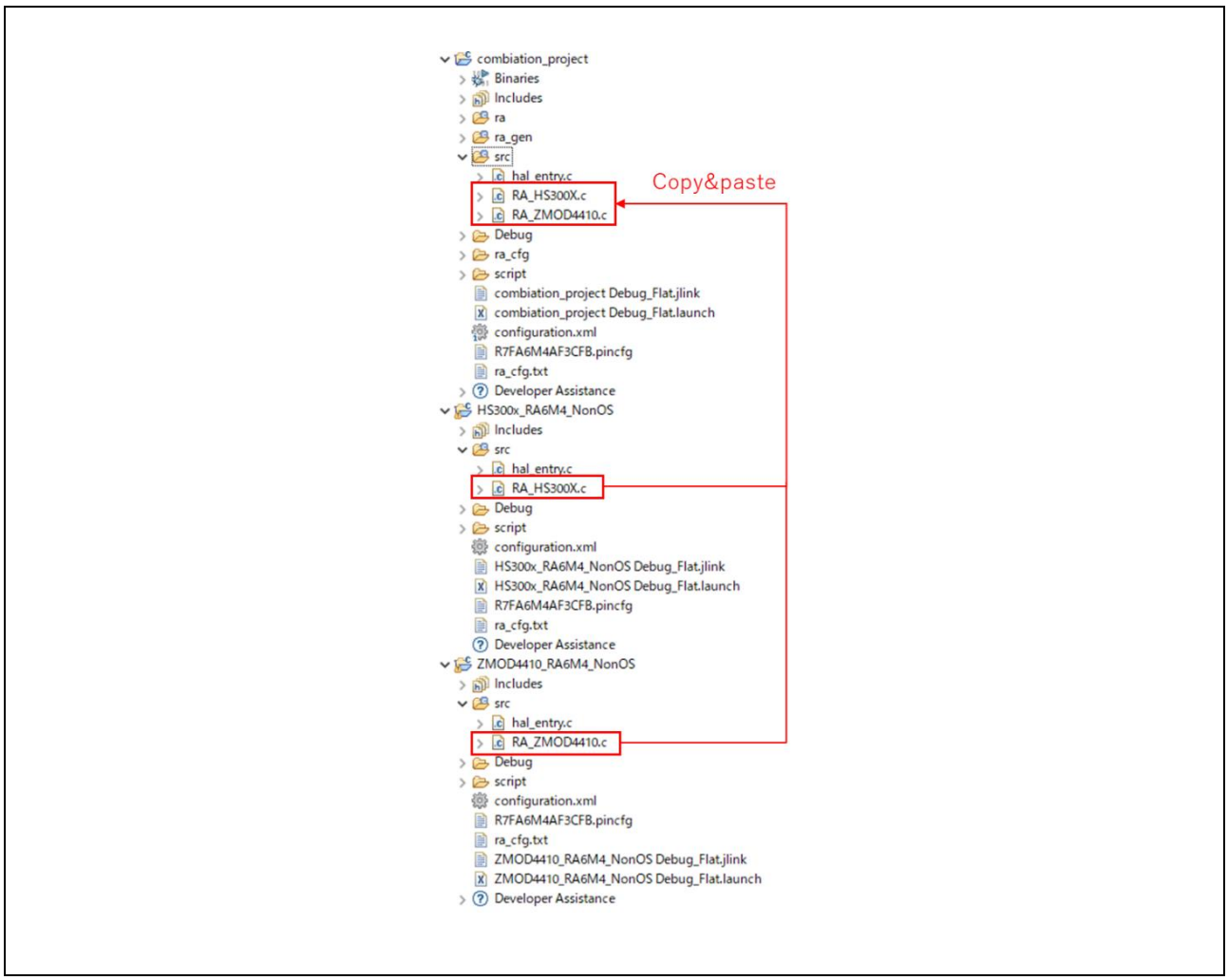
The code change procedures are explained separately for NonOS, FreeRTOS, and Azure.

3.1 NonOS

Note: If the version of the NonOS project for ZMOD4410 that is used in combination with other sensor software sample projects is v1.52 to v1.53, switch to the latest version.

3.1.1 Copy and paste files

Copy and paste c file in which the application is described from each sample project.



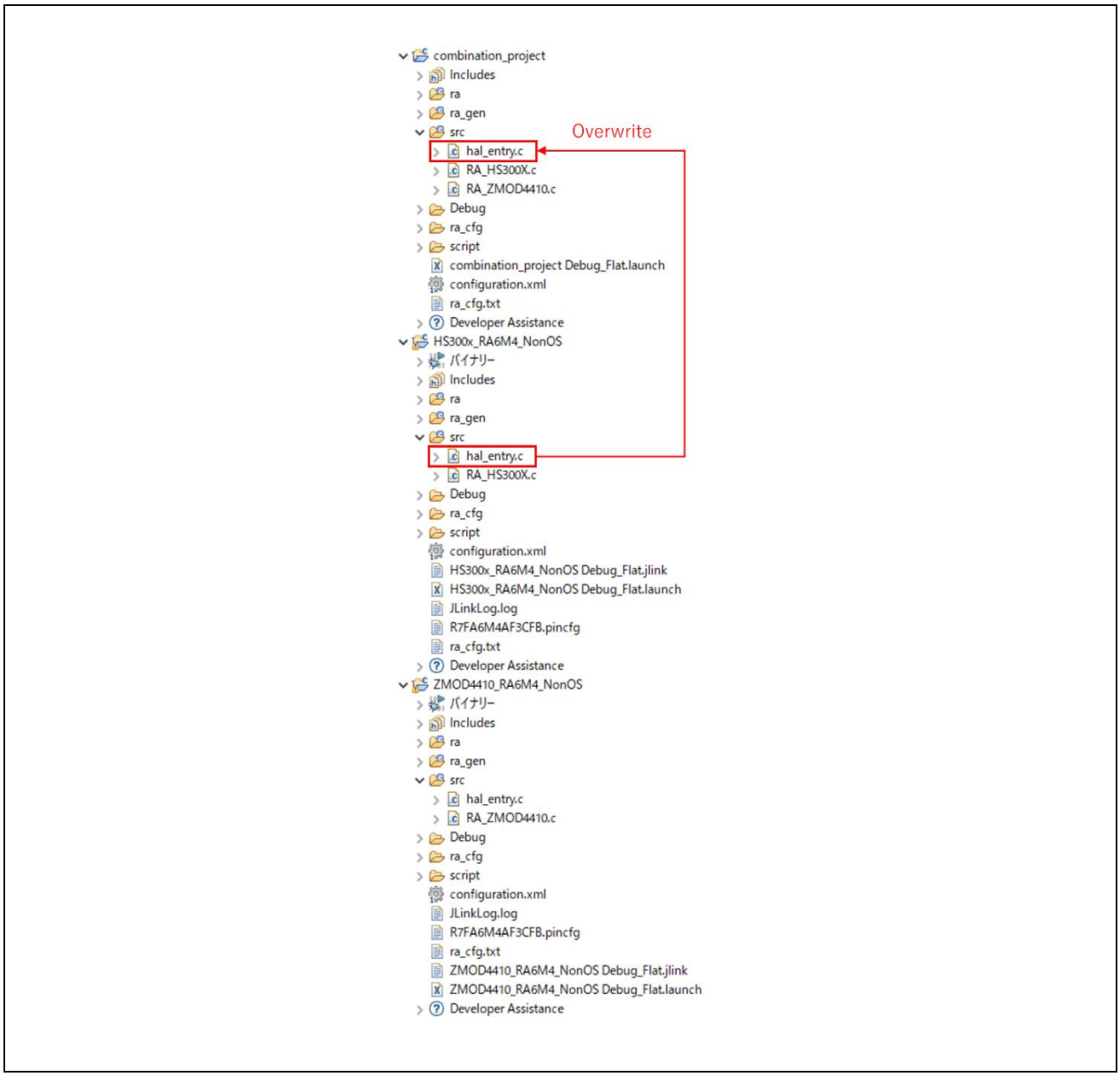
For OB1203 sensor, ob1203_bio folder must also be copied and pasted.

For RL78G14 MCU, copy and paste application, general, r_bsp, r_comms_i2c_rl, r_config, r_(sensor_name) folder from each sensor software projects.

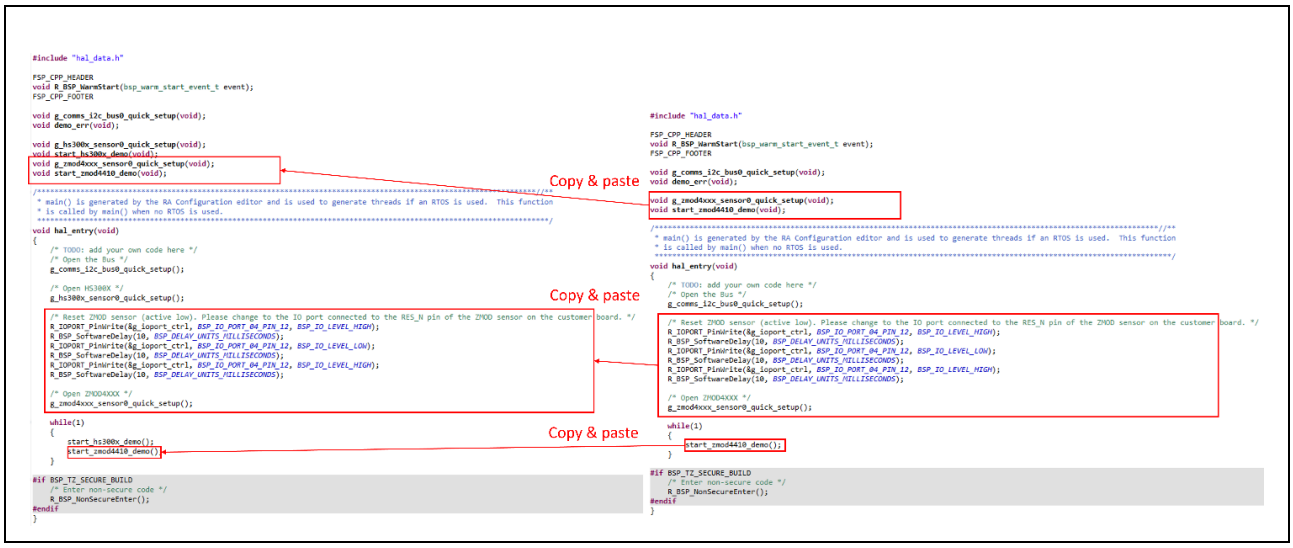
If the folder or file that the same name exists, overwrite the folder or file.

3.1.2 Add initialization process and main process

Overwrite the C file where main() or hal_entry() is declared from the sample project.



Also, copy and paste the sensor initialization process, main process, and the prototype declaration from The C file that main() or hal_entry() is declared.



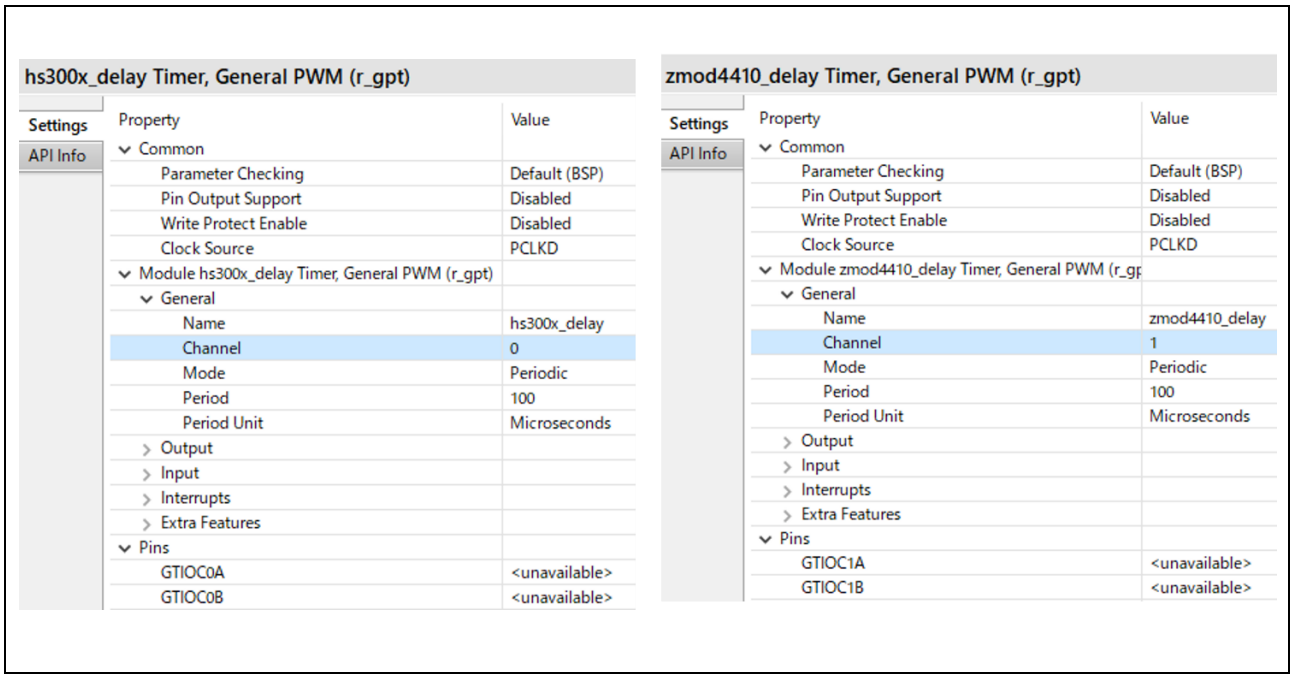
When using the RX project for ZMOD4xxx, add definition of including r_gpio_rx_if.h.

3.1.3 Timer module settings

The timer module used by each application code sets a different channel.

3.1.3.1 RA, RZ

Set as follows in FPS Configurator.



3.1.3.2 RL78G14

Set timers for different channels in the peripheral functions of Code Generator.

3.1.3.3 RX, RL78/G2X

Add timer components for different channels by “add component” in Smart Configurator.

3.1.4 Timer module API (RX, RL78/G14, RL78/G2X Only)

Change the timer module API in the application code to correspond to the set channel.

3.1.4.1 RX

Before

```
static void zmod4410_delay_start(uint32_t delay, zmod4410_delay_units_t units)
{
    /* Convert to units of ZMOD4410_DELAY_PERIOD */
    gs_zmod4410_delay_count = (delay * gs_zmod4410_delay_time[units]) / ZMOD4410_DELAY_PERIOD;

    /* Stop timer */
    R_Config_CMT0_Stop();

    /* Reset counter */
    R_Config_CMT0_Reset();

    /* Start timer */
    R_Config_CMT0_Start();
}

static bool zmod4410_delay_wait(void)
{
    bool wait;

    if (gs_zmod4410_delay_count > 0)
    {
        wait = true;
    }
    else
    {
        /* Stop timer */
        R_Config_CMT0_Stop();
        wait = false;
    }
    return wait;
}
```

After

```
static void zmod4410_delay_start(uint32_t delay, zmod4410_delay_units_t units)
{
    /* Convert to units of ZMOD4410_DELAY_PERIOD */
    gs_zmod4410_delay_count = (delay * gs_zmod4410_delay_time[units]) / ZMOD4410_DELAY_PERIOD;

    /* Stop timer */
    R_Config_CMT1_Stop();

    /* Reset counter */
    R_Config_CMT1_Reset();

    /* Start timer */
    R_Config_CMT1_Start();
}

static bool zmod4410_delay_wait(void)
{
    bool wait;

    if (gs_zmod4410_delay_count > 0)
    {
        wait = true;
    }
    else
    {
        /* Stop timer */
        R_Config_CMT1_Stop();
        wait = false;
    }
    return wait;
}
```

Since R_Config_CMT0_Reset() used in the sample project for RX is a user-defined function, it must be newly defined in the c file of the timer module when combining.

Define the function as follows in the user code description part of the c file.

```
void R_Config_CMT1_Reset(void)
{
        /* Reset counter */
        CMT1.CMCNT = 0x0000;
}
```

Add the prototype declaration of R_Config_CMT0_Reset() to the user code description part in h file of the timer module.

3.1.4.2 RL78/G14

Before

```
static void zmod4410_delay_start(uint32_t delay, zmod4410_delay_units_t units)
{
    /* Convert to units of ZMOD4410_DELAY_PERIOD */
    gs_zmod4410_delay_count = (delay * gs_zmod4410_delay_time[units]) / ZMOD4410_DELAY_PERIOD;

    /* Stop timer */
    R_TAU0_Channel0_Stop();

    /* Reset counter */
    R_TAU0_Channel0_Reset();

    /* Start timer */
    R_TAU0_Channel0_Start();
}

static bool zmod4410_delay_wait(void)
{
    bool wait;

    if (gs_zmod4410_delay_count > 0)
    {
        wait = true;
    }
    else
    {
        /* Stop timer */
        R_TAU0_Channel0_Stop();
        wait = false;
    }

    return wait;
}
```

After

```
static void zmod4410_delay_start(uint32_t delay, zmod4410_delay_units_t units)
{
    /* Convert to units of ZMOD4410_DELAY_PERIOD */
    gs_zmod4410_delay_count = (delay * gs_zmod4410_delay_time[units]) / ZMOD4410_DELAY_PERIOD;

    /* Stop timer */
    R_TAU0_Channel1_Stop();

    /* Reset counter */
    R_TAU0_Channel1_Reset();

    /* Start timer */
    R_TAU0_Channel1_Start();
}

static bool zmod4410_delay_wait(void)
{
    bool wait;

    if (gs_zmod4410_delay_count > 0)
    {
        wait = true;
    }
    else
    {
        /* Stop timer */
        R_TAU0_Channel1_Stop();
        wait = false;
    }

    return wait;
}
```

Since R_TAU0_Channel0_Reset() used in the sample project for RL78G14 is a user-defined function, it must be newly defined in the c file of the timer module when combining.

Define the function as follows in the user code description part of the c file.

```
void R_TAU0_Channel1_Reset(void)
{
        /* function not supported by this module */
}
```

Add the prototype declaration of R_TAU0_Channel0_Reset() to the user code description part in h file of the timer module.

3.1.4.3 RL78/G23

Before

```
static void zmod4410_delay_start(uint32_t delay, zmod4410_delay_units_t units)
{
    /* Convert to units of ZMOD4410_DELAY_PERIOD */
    gs_zmod4410_delay_count = (delay * gs_zmod4410_delay_time[units]) / ZMOD4410_DELAY_PERIOD;

    /* Stop timer */
    R_Config_TAU0_0_Stop();

    /* Reset counter */
    R_Config_TAU0_0_Reset();

    /* Start timer */
    R_Config_TAU0_0_Start();
}

static bool zmod4410_delay_wait(void)
{
    bool wait;

    if (gs_zmod4410_delay_count > 0)
    {
        wait = true;
    }
    else
    {
        /* Stop timer */
        R_Config_TAU0_0_Stop();
        wait = false;
    }

    return wait;
}
```

After

```
static void zmod4410_delay_start(uint32_t delay, zmod4410_delay_units_t units)
{
    /* Convert to units of ZMOD4410_DELAY_PERIOD */
    gs_zmod4410_delay_count = (delay * gs_zmod4410_delay_time[units]) / ZMOD4410_DELAY_PERIOD;

    /* Stop timer */
    R_Config_TAU0_1_Stop();

    /* Reset counter */
    R_Config_TAU0_1_Reset();

    /* Start timer */
    R_Config_TAU0_1_Start();
}

static bool zmod4410_delay_wait(void)
{
    bool wait;

    if (gs_zmod4410_delay_count > 0)
    {
        wait = true;
    }
    else
    {
        /* Stop timer */
        R_Config_TAU0_1_Stop();
        wait = false;
    }

    return wait;
}
```

Since R_Config_TAU0_0_Reset() used in the sample project for RL78G23 is a user-defined function, it must be newly defined in the c file of the timer module when combining.

Define the function as follows in the user code description part of the c file.

```
void R_Config_TAU0_1_Reset(void)
{
        /* function not supported by this module */
}
```

Add the prototype declaration of R_Config_TAU0_0_Reset() to the user code description part in h file of the timer module.

3.1.5 Add definition of including

When using ZMOD4xxx sensors and using IRQ, add definition of including in platform.h as follow.

```
#include "r_cg_intc.h"
```

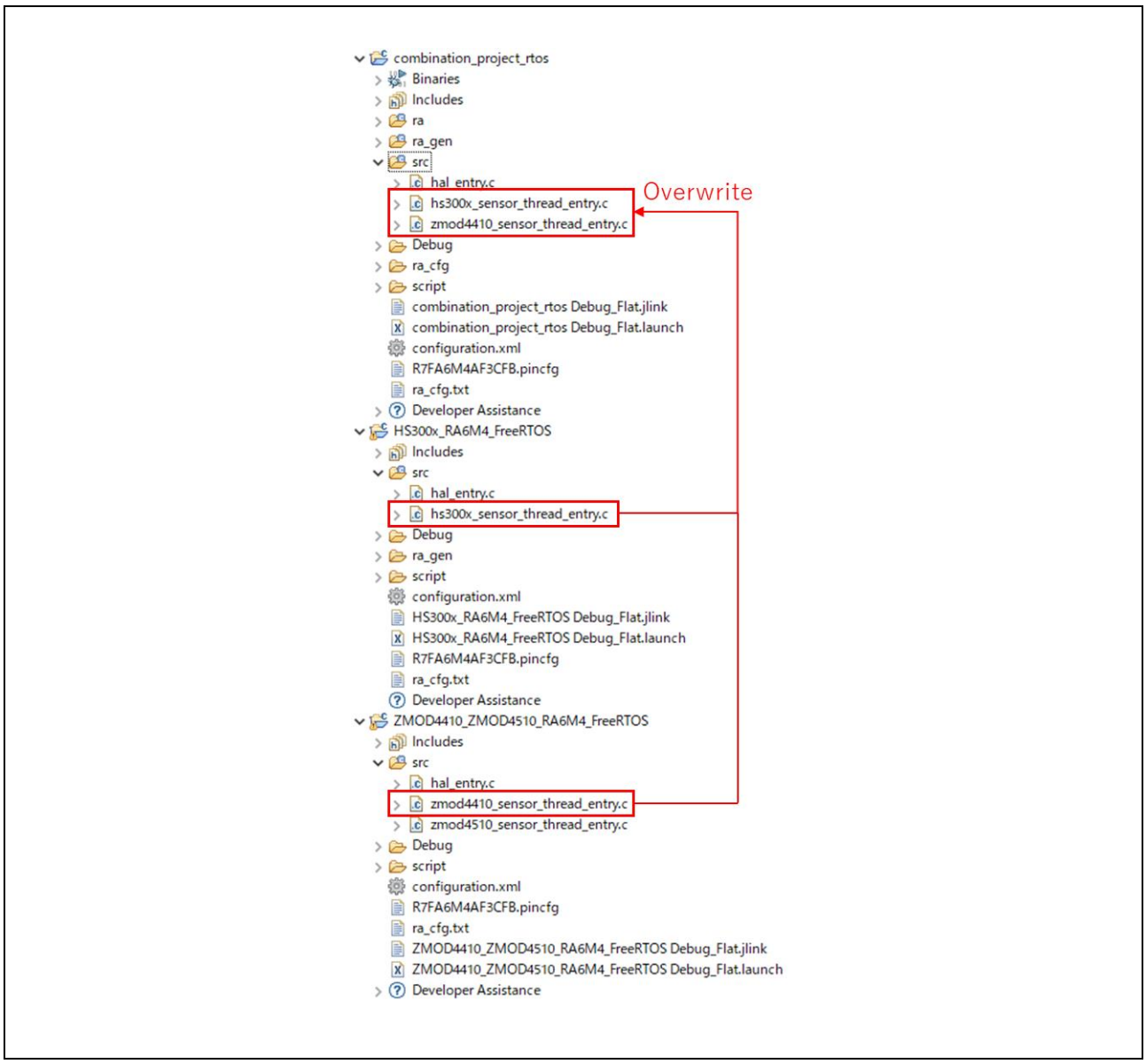
When using OB1203 sensor, add definition of including in r_smc_entry.h as follow.

```
#include "r_cg_timer.h"
```

3.2 FreeRTOS

3.2.1 Overwrite files

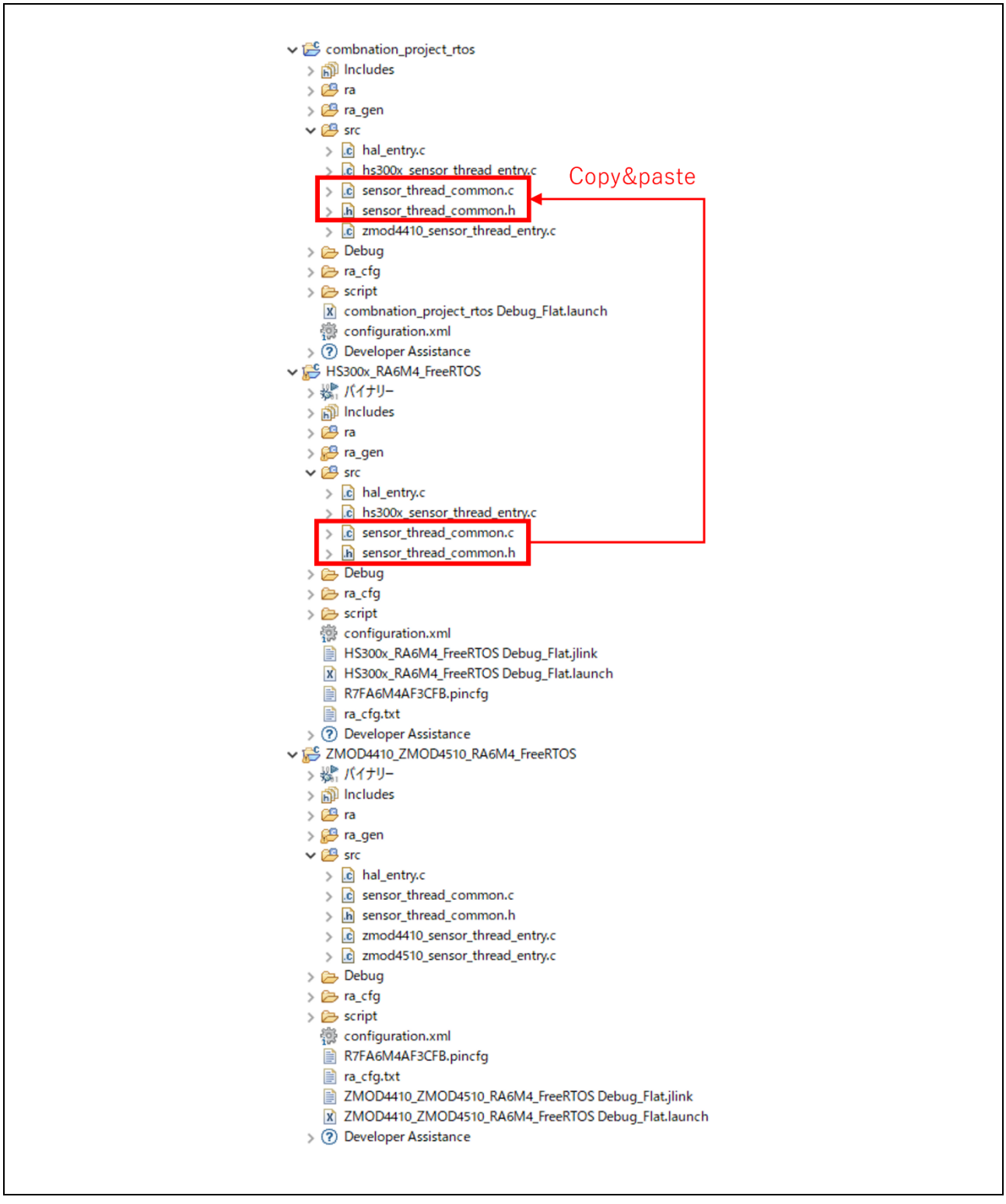
Overwrite c file in which the application is described from each sample project.



For OB1203 sensor, ob1203_bio folder must also be copied and pasted.

3.2.2 Common function (RA, RZ Only)

Copy and paste the sensor_thread_common.c file and sensor_thread_common.h file that describe common functions from the sample project.



3.2.3 Add initialization process and main process (RX Only)

When using the RX FreeRTOS project, refer to 3.2.1 Add initialization process and main process.

3.2.4 Sampling period

Depending on the sensor, it may be necessary to read from the sensor at a specific sampling period. In such cases, add vTaskDelay() for switching to another thread or changing the priority of threads in order to comply with the sampling period.

3.2.5 Enable sensor reset processing (RA, RZ, ZMOD4xxx sensor only)

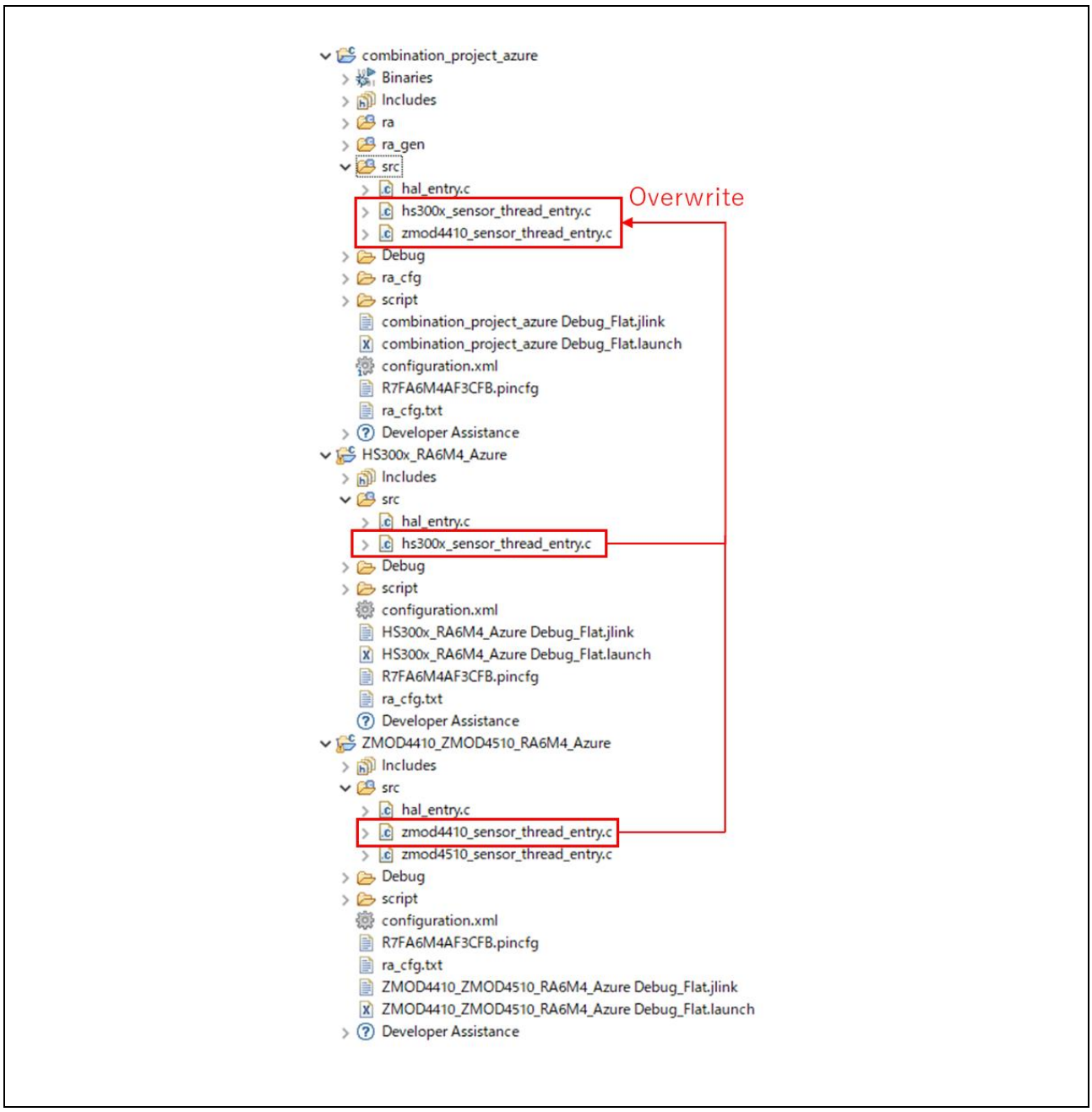
Set the value of “G_ZMOD4XXX_SENSOR_RESET_ENABLE” to 1 when using ZMOD4xxx sensors in combination.

However, when using multiple ZMOD4xxx sensors in combination, set the value of "G_ZMOD4XXX_SENSOR_RESET_ENABLE" defined in the first sensor thread to 1, and set it to 0 in other threads. Basically, the first created thread is called first.

3.3 Azure

3.3.1 Overwrite files

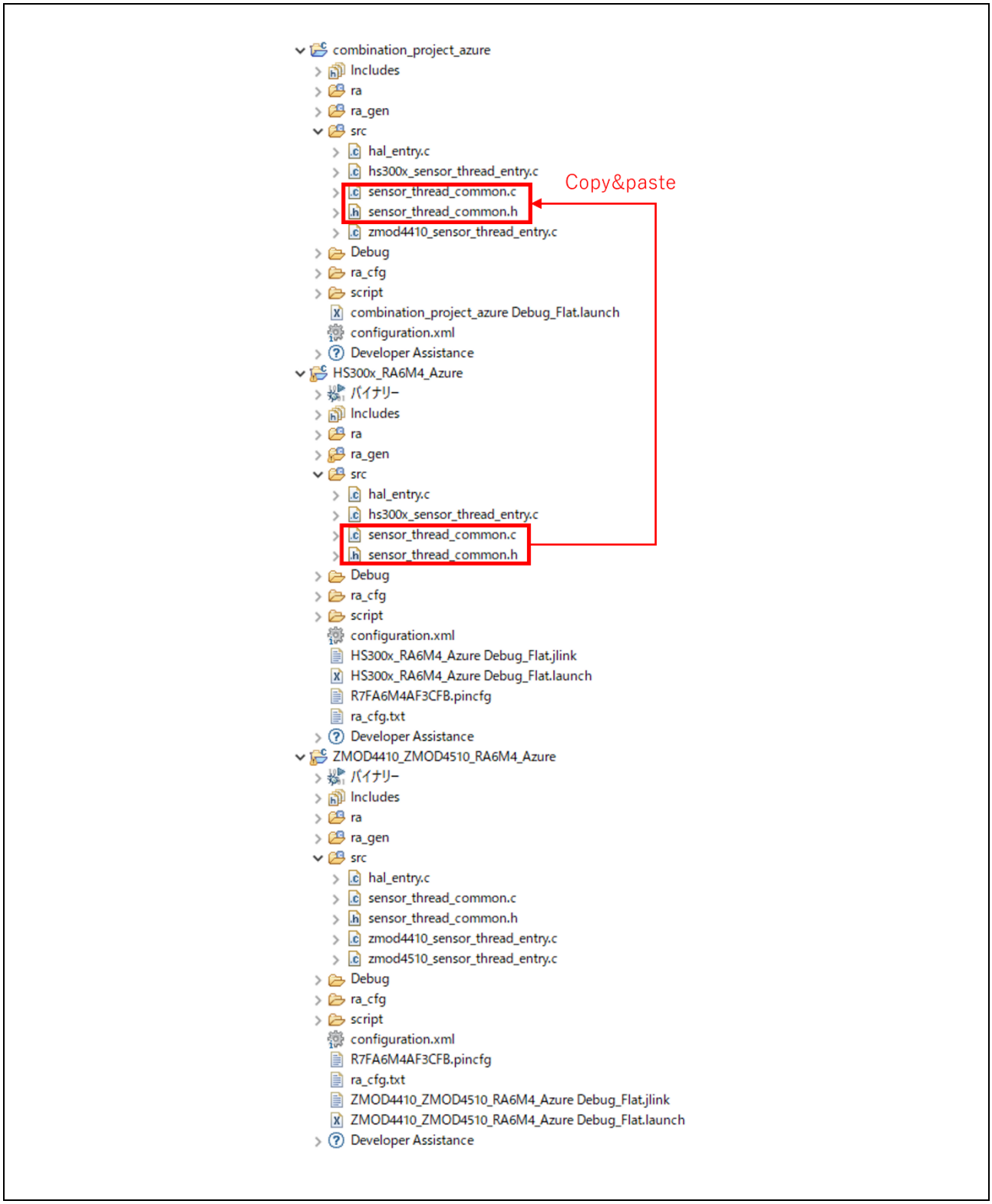
Overwrite c file in which the application is described from each sample project.



For OB1203 sensor, ob1203_bio folder must also be copied and pasted.

3.3.2 Common function

Copy and paste the sensor_thread_common.c file and sensor_thread_common.h file that describe common functions from the sample project.



3.3.3 Sampling period

Depending on the sensor, it may be necessary to read from the sensor at a specific sampling period. In such cases, add tx_thread_sleep() for switching to another thread or changing the priority of threads in order to comply with the sampling period.

3.3.4 Enable sensor reset processing (ZMOD4xxx sensor only)

Set the value of “G_ZMOD4XXX_SENSOR_RESET_ENABLE” to 1 when using ZMOD4xxx sensors in combination.

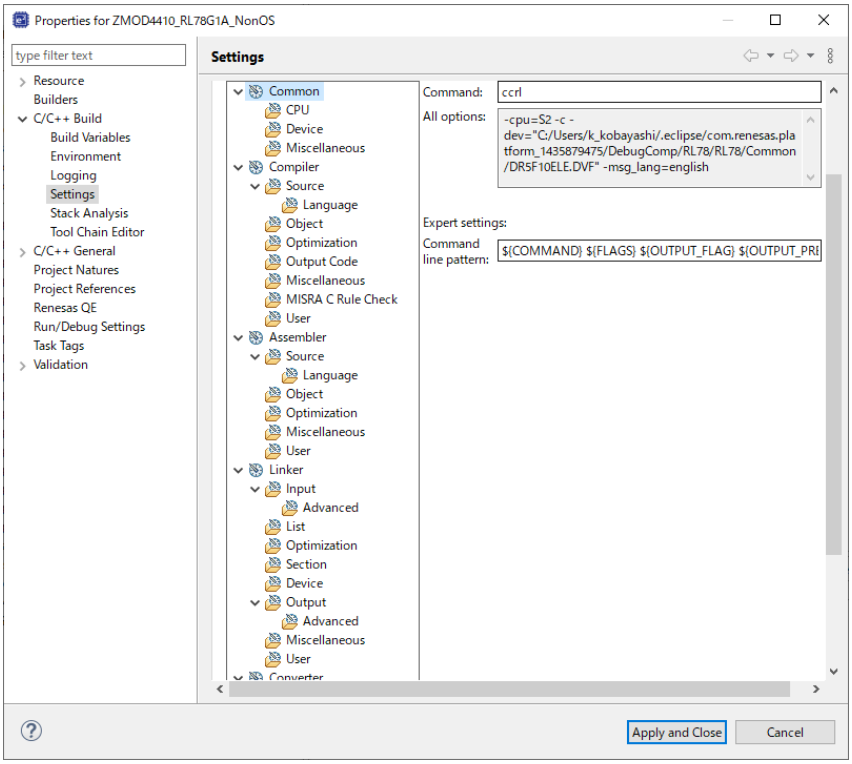
However, when using multiple ZMOD4xxx sensors in combination, set the value of "G_ZMOD4XXX_SENSOR_RESET_ENABLE" defined in the first sensor thread to 1, and set it to 0 in other threads. Basically, the first created thread is called first.

4. Project settings

4.1 RL78/G14

Open the "Properties" window for the project.

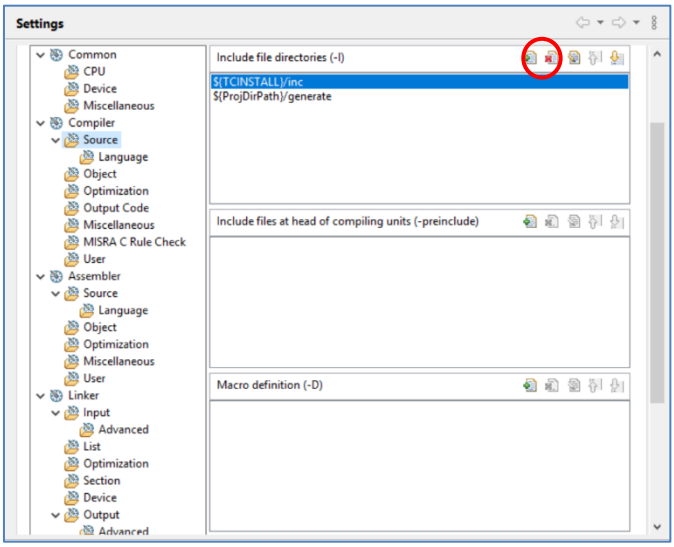
Select [C/C++ Build] → [Settings] in the "Properties" window to open the "Settings" panel.

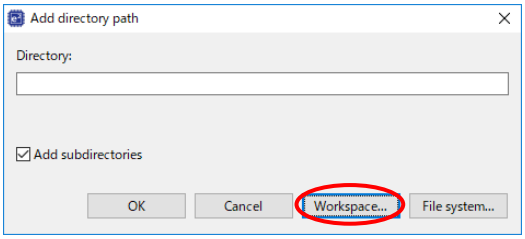


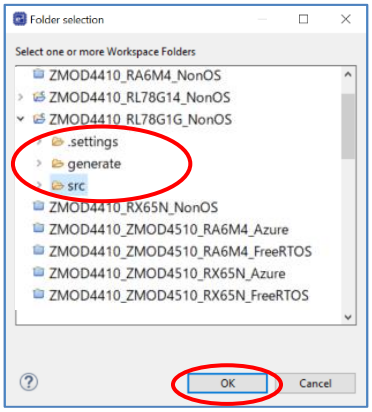
Select [Compiler] → [Source] in the "Tool Settings" tabbed page and press the [Add] icon.

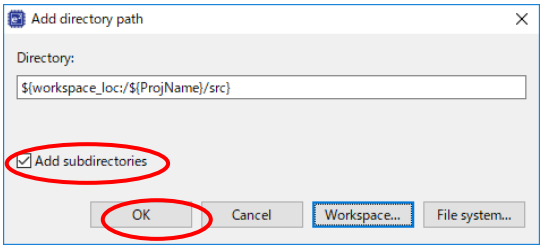
Press the [Workspace] button in the [Add directory path] dialog box and a list of projects will appear. Select the "src" folder for the newly created project in the list and press the [OK] button.

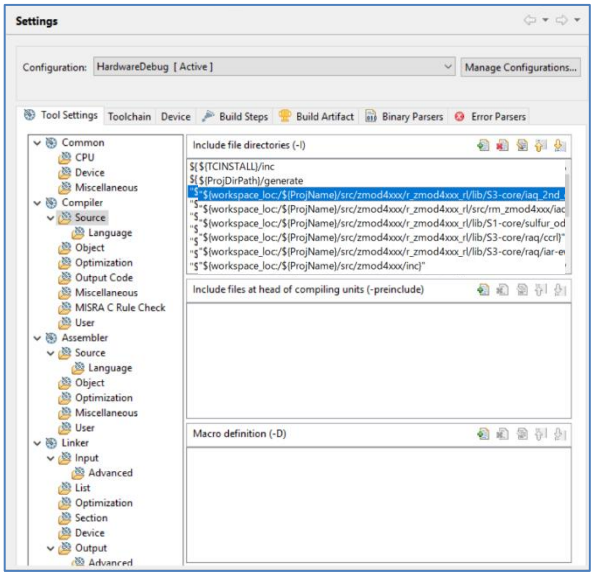
Select the checkbox for "Add subdirectories" and press the [OK] button.





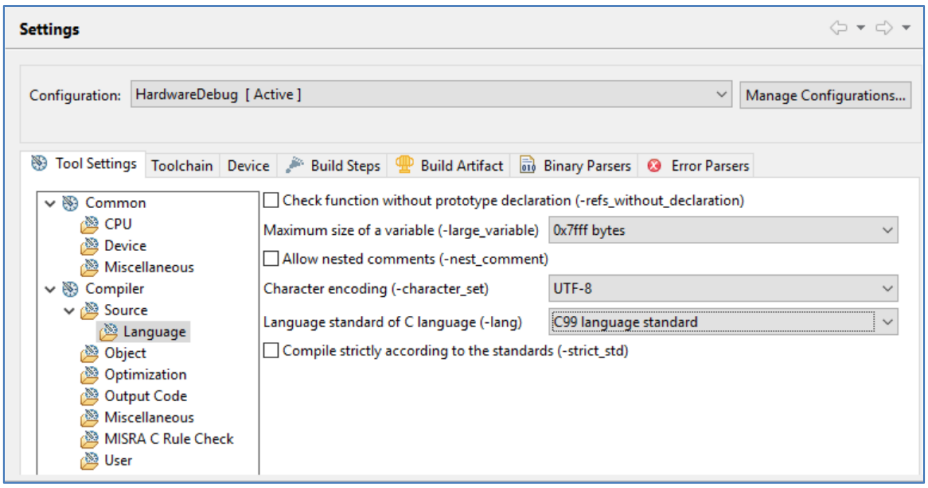






Select [Compiler] → [Source] → [Language] in the "Tool Settings" tabbed page and change the setting of "Language standard of C language" to "C99 language standard".

Press the [Apply and Close] button to close the "Properties" window.

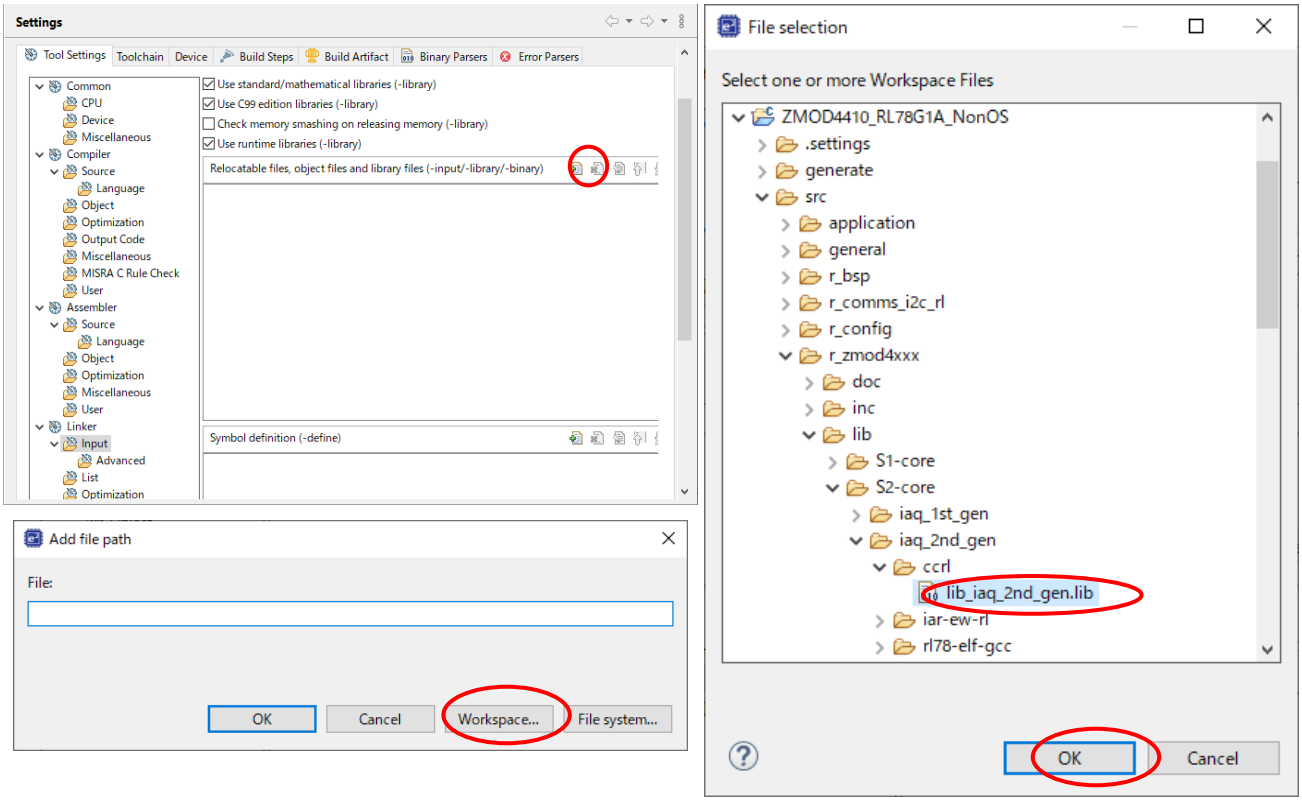


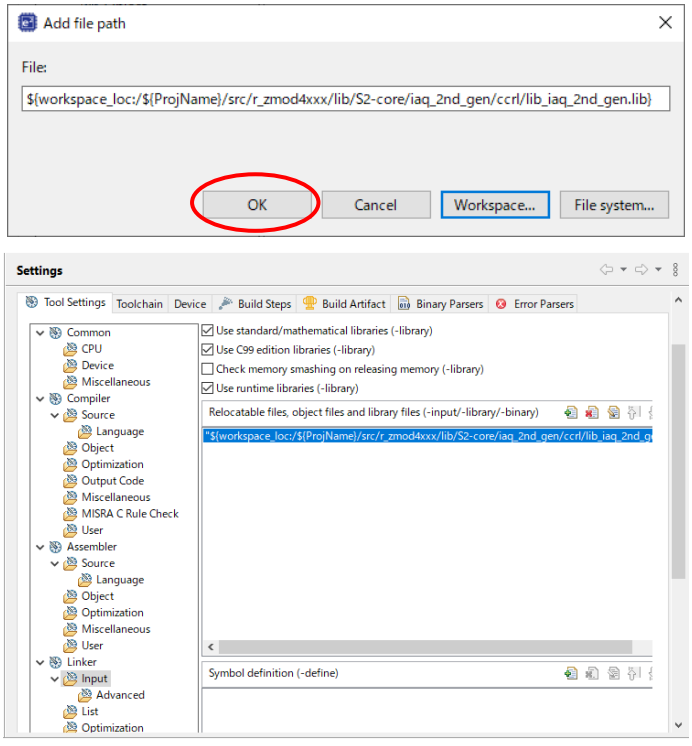
When using ZMOD4xxx sensors, [Linker] setting must be modified as follow.

Select [Linker] → [Input] in the "Tool Settings" tabbed page and press the [Add] icon.

Press the [Workspace] button in the [Add directory path] dialog box and a list of projects will appear.

Select the "src" folder for the newly created project in the list and select the lib file that according to MCU architecture, measurement mode, and compiler to be used from the "r_zmod4xxx/lib" folder.

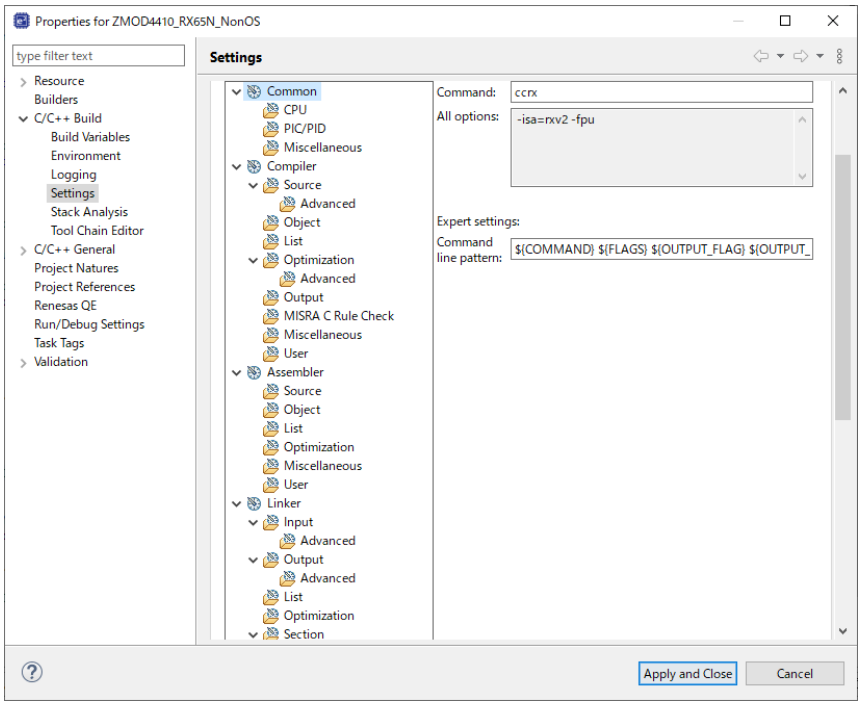




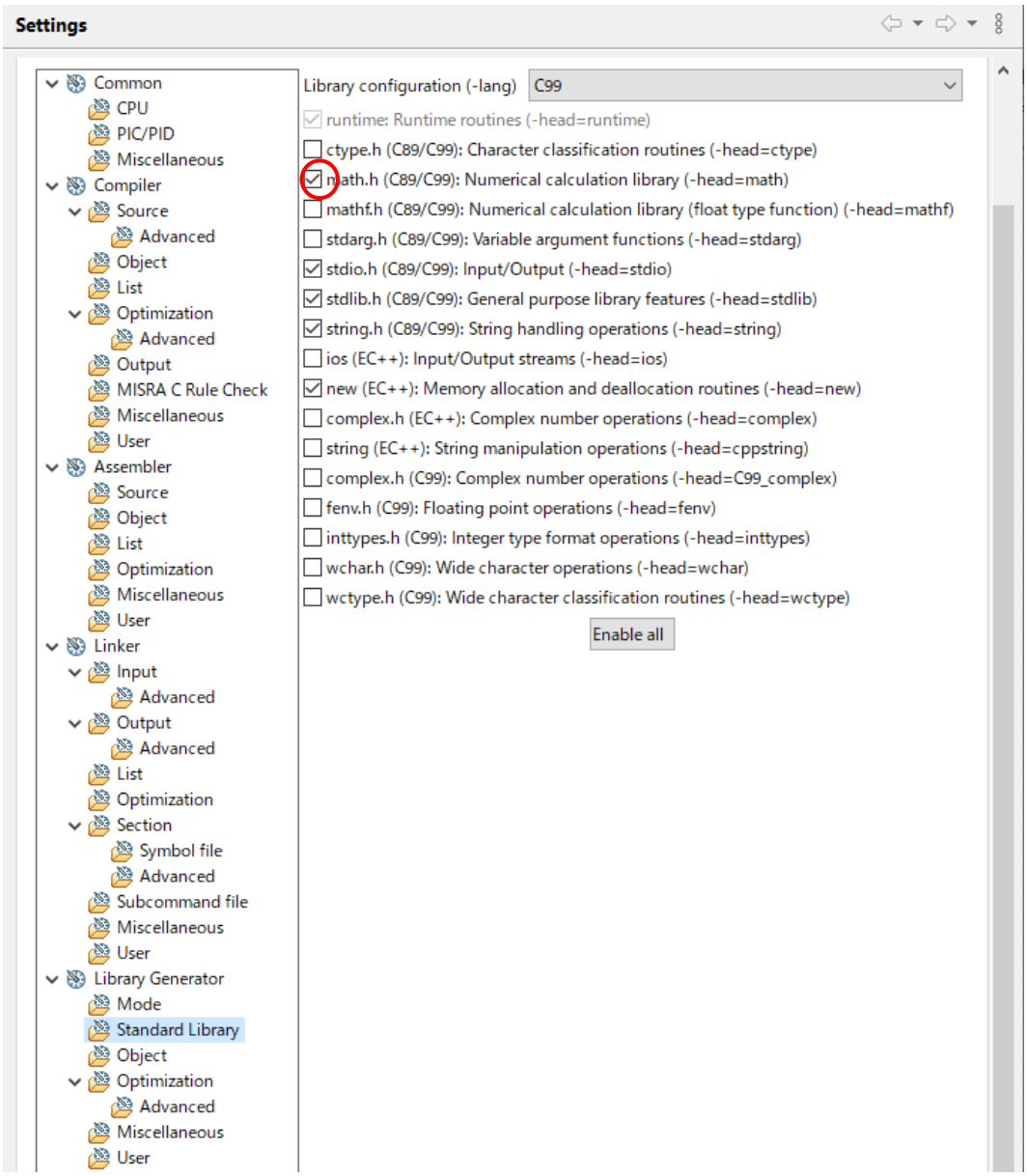
4.2 RX (ZMOD4xxx sensor only)

Open the "Properties" window for the project.

Select [C/C++ Build] → [Settings] in the "Properties" window to open the "Settings" panel.



Select [Library Generator] → [Standard Library] in the "Tool Settings" tabbed page and enable "math.h (C89/99)".



5. Appendix

5.1 API processing period

Processing periods of APIs are explained using ZMOD4410 IAQ 2nd Gen as an example.

item	content
Board	RTK7EKA6M4S00001BE (EK-RA6M4)
MCU	RA6M4 (R7FA6M4AF3CFB :144pin)
Clock	200MHz
Voltage	5V
Environment	e2 Studio 2022-07
C compiler	GCC 10.3.1.20210824
FSP	V.4.0.0

API	Period
RM_ZMOD4XXX_Open	233ms
RM_ZMOD4XXX_MeasurementStart	2.5us
RM_ZMOD4XXX_Read	2.5us
RM_ZMOD4XXX_Iaq2ndGenDataCalculate	3.2us (in stabilization) 637us (after stabilization is complete)
RM_ZMOD4XXX_StatusCheck	3us
RM_ZMOD4XXX_DeviceErrorCheck	2.5us

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep 30, 2022	-	First Release
1.10	Dec 27, 2022	p5	add description of RL78/G14
1.20	Sep 7, 2023	-	Added description of I2C Shared Bus Removed, added and updated description of Code change procedure

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

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

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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