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
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 .................................................................................................................................... 3
2. I2S Shared Bus ......................................................................................................................... 3
2.1 RA, RZ ..................................................................................................................................................... 4
2.2 RX, RL78/G2x ......................................................................................................................................... 5
3. Code change procedure ............................................................................................................ 6
3.1 NonOS ..................................................................................................................................................... 7
3.1.1 Copy and paste files .............................................................................................................................. 7
3.1.2 Rename start_demo() ........................................................................................................................... 8
### 3.1.3 Change to global variables
- Page 8

### 3.1.4 Remove initialization process
- Page 9

### 3.1.5 Change while loop
- Page 10

### 3.1.6 Add process for sensor change
- Page 11

### 3.1.7 Common functions
- Page 12

### 3.1.8 Add initialization process and main process
- Page 12

### 3.1.9 Sampling period
- Page 12

### 3.1.10 Change in processing of waiting for measurement completion (OB1203 sensor only)
- Page 13

### 3.2 FreeRTOS
- Page 16

#### 3.2.1 Overwrite files
- Page 16

#### 3.2.2 Add vTaskDelay()
- Page 17

#### 3.2.3 Common function
- Page 18

#### 3.2.4 Sampling period
- Page 18

#### 3.2.5 Change in processing of waiting for measurement completion (OB1203 sensor only)
- Page 19

### 3.3 Azure
- Page 21

#### 3.3.1 Overwrite files
- Page 21

#### 3.3.2 Add tx_thread_sleep()
- Page 22

#### 3.3.3 Common function
- Page 23

#### 3.3.4 Sampling period
- Page 23

#### 3.3.5 Change in processing of waiting for measurement completion (OB1203 sensor only)
- Page 24

### 4. Appendix
- Page 26

#### 4.1.1 API processing period
- Page 26

### Revision History
- Page 27

### General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products
- Page 28

### Notice
- Page 29

### Corporate Headquarters
- Page 29

### Contact information
- Page 29

### Trademarks
- Page 29
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. **I2S 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
2.2 RX, RL78/G2x

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

<table>
<thead>
<tr>
<th>The same I2C bus is shared</th>
<th>The different I2C bus is used</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Next, set the “I2C Shared Bus No.” for the I2C Communication Device assigned to each sensor.

- 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 a different I2C Shared Bus number

<table>
<thead>
<tr>
<th>The same I2C bus is shared</th>
<th>The different I2C bus is used</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3.png" alt="Code Snippet" /></td>
<td><img src="image4.png" alt="Code Snippet" /></td>
</tr>
</tbody>
</table>
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 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.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>const unsigned long long bsp_delay_time[] = {</td>
<td>extern const unsigned long long bsp_delay_time[];</td>
</tr>
<tr>
<td>1,</td>
<td></td>
</tr>
<tr>
<td>1000,</td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td></td>
</tr>
<tr>
<td>};</td>
<td></td>
</tr>
</tbody>
</table>

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

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.
3.1.2 Rename start_demo()
Rename start_demo() defined in c file for each sensor to start_(sensor name)_demo().

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void start_demo(void)</code></td>
<td><code>void start_hs3001_demo(void)</code></td>
</tr>
</tbody>
</table>

3.1.3 Change to global variables
Change the local variables “raw_data” and “sequence” defined in start_(sensor name)_demo() to static global variables.

In the case of OB1203 sensor, in addition to “raw_data” and “sequence”, seven other local variables must be changed to static global variables.
### 3.1.4 Remove initialization process

Remove processing from the local variable declaration to while loop.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>void start_hs3001_demo(void)</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>fsp_err_t err;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>/* Open the Bus */</td>
<td></td>
</tr>
<tr>
<td>g_comms_i2c_bus0_quick_setup();</td>
<td></td>
</tr>
<tr>
<td>/* Open HS300X */</td>
<td></td>
</tr>
<tr>
<td>g_hs300x_sensor0_quick_setup();</td>
<td></td>
</tr>
<tr>
<td>while(1)</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>switch(sequence)</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>void start_hs3001_demo(void)</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>fsp_err_t err;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>while(1)</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>switch(sequence)</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

For OB1203 sensor, before removing the initialization process, it is necessary to create an initialization function for OB1203 sensor and move the OB1203 initialization process. After moving, remove the initialization process.
### 3.1.5 Change while loop

Add “sensor_change” flag and change the while loop to the do-while statement.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
</table>
| ```c
void start_hs3001_demo(void)
{
    fsp_err_t err;
    while(1)
    {
        switch(sequence)
        ...
    }
}
``` | ```c
void start_hs3001_demo(void)
{
    fsp_err_t err;
    bool sensor_change = false;
    do
    {
        switch(sequence)
        ...
    } while(!sensor_change);
}
``` |

For OB1203 sensor, remove the while loop.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
</table>
| ```c
void start_ob1203_demo(void)
{
    bool result;
    while(1)
    {
        switch (sequence)
        ...
    }
}
``` | ```c
void start_ob1203_demo(void)
{
    bool result;
    switch (sequence)
    ...
}
``` |
3.1.6 Add process for sensor change

Add “sensor_change = true;” to each process at the completion of I2C communication described in 
start_(sensor name)_demo().

```c
switch (gs_demo_callback_status)
{
    case DEMO_CALLBACK_STATUS_WAIT :
        break;
    case DEMO_CALLBACK_STATUS_SUCCESS :
        sensor_change = true;
        sequence = DEMO_SEQUENCE_3;
        break;
    case DEMO_CALLBACK_STATUS_REPEAT :
        sequence = DEMO_SEQUENCE_1;
        break;
    default :
        demo_err();
        break;
}
```

If an IRQ signal is used, also add “sensor_change = true;” to the process when receiving the IRQ signal.

```c
switch (gs_irq_callback_status)
{
    case DEMO_CALLBACK_STATUS_WAIT :
        break;
    case DEMO_CALLBACK_STATUS_SUCCESS :
        gs_irq_callback_status = DEMO_CALLBACK_STATUS_WAIT;
        if (((RM_ZMOD4410_LIB_TYPE_IAQ_2ND_GEN == lib_type) ||
            (RM_ZMOD4410_LIB_TYPE_IAQ_2ND_GEN_ULP == lib_type))
            {
            sequence = DEMO_SEQUENCE_5;
        }
        else
        {
            sequence = DEMO_SEQUENCE_7;
        }
        break;
    default :
        demo_err();
        break;
}
```

For OB1203 sensor, this change is not necessary.
3.1.7 Common functions
Move the I2C driver initialization function `g_comms_i2c_bus0_quick_setup()` and `demo_err()` described in the c file for each sensor to the c file where `main()` or `hal_entry()` is declared. After moving, remove duplicate functions.

3.1.8 Add initialization process and main process
Describe each initialization process in `main()` or `hal_entry()` and each main process in the while loop.

3.1.9 Sampling period
Depending on the sensor, it may be necessary to read from the sensor at a specific sampling period. In such cases, change `SoftwareDelay()` to a timer module in order to comply with the sampling period.
3.1.10 Change in processing of waiting for measurement completion (OB1203 sensor only)

Only when using OB1203 sensor, the measurement completion wait processing needs to be changed.

First, add the global variable "complete" as a measurement completion flag in (MCU name)_OB1203.c

```c
static rm_ob1203_raw_data_t raw_data;
static rm_ob1203_ppg_data_t ppg_data;
static ob1203_bio_t ob1203_bio;
static bool change = false;
static bool valid = false;
static bool update = false;
static bool ready = false;
static bool complete = false;
static ob1203_bio_gain_currents_t gain_currents;
static demo_sequence_t sequence = DEMO_SEQUENCE_1;
```

Then, change the measurement completion wait processing in (MCU name)_OB1203.c as follows.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>case DEMO_SEQUENCE_3:</td>
<td>case DEMO_SEQUENCE_3:</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>/* Wait measurement period */</td>
<td>/* Wait measurement period */</td>
</tr>
<tr>
<td>result = ob1203_bio_measurement_period_wait(&amp;ob1203_bio);</td>
<td>result = ob1203_bio_measurement_period_wait(&amp;ob1203_bio, complete);</td>
</tr>
<tr>
<td>if (false == result)</td>
<td>if (false == result)</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>demo_err();</td>
<td>demo_err();</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td>if (complete)</td>
</tr>
<tr>
<td>sequence = DEMO_SEQUENCE_4;</td>
<td>{</td>
</tr>
<tr>
<td>break;</td>
<td>sequence = DEMO_SEQUENCE_4;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Go to ob1203_bio.c and change the argument of ob1203_bio_measurement_period_wait() as follows.

At this time, change the description of the prototype declaration as well.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool ob1203_bio_measurement_period_wait(ob1203_bio_t *const p_bio)</td>
<td>bool ob1203_bio_measurement_period_wait(ob1203_bio_t *const p_bio, bool *const p_complete)</td>
</tr>
</tbody>
</table>
### 3.1.10.1 With IRQ signal

Change the description in `ob1203_bio_measurement_period_wait()` as follows.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>/* Wait IRQ callback */</td>
<td>/* Wait IRQ callback */</td>
</tr>
<tr>
<td>ob1203_bio_irq_callback_wait();</td>
<td>ob1203_bio_irq_callback_wait(p_complete);</td>
</tr>
<tr>
<td>/* Clear the IRQ callback flag */</td>
<td>if (*p_complete)</td>
</tr>
<tr>
<td>ob1203_bio_irq_callback_flag_clear();</td>
<td>/* Clear the IRQ callback flag */</td>
</tr>
<tr>
<td></td>
<td>ob1203_bio_irq_callback_flag_clear();</td>
</tr>
</tbody>
</table>

Next, change `ob1203_bio_irq_callback_wait()` as follows.

At this time, change the description of the prototype declaration as well.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void ob1203_bio_irq_callback_wait(void)</td>
<td></td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>while (OB1203_BIO_CALLBACK_STATUS_WAIT == gs_irq_callback_status)</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td>if (OB1203_BIO_CALLBACK_STATUS_WAIT == gs_irq_callback_status)</td>
</tr>
<tr>
<td>/* nothing */</td>
<td>*p_complete = false;</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>*p_complete = true;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
### 3.10.2 Without IRQ signal

Change the description in `ob1203_bio_measurement_period_wait()` as follows.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
</table>
| ```c
    do
    /* Get Device status */
    counter = OB1203_BIO_MAX_COUNTS;
    do
    { ...
    } while ((false == ob1203_bio_i2c_communication_success()) &&
              (0 != counter));

    if (0 == counter)
    { return false;
    }

    while(false == *p_flag);
``` |
| ```c
    do
    /* Get Device status */
    counter = OB1203_BIO_MAX_COUNTS;
    do
    { ...
    } while ((false == ob1203_bio_i2c_communication_success()) &&
              (0 != counter));

    if (0 == counter)
    { return false;
    }

    *p_complete = *p_flag;
``` |
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 Add vTaskDelay()

Add `vTaskDelay()` for switching another thread to (sensor name)_thread_entry() of each sensor.

**Before**

```c
void hs300x_sensor_thread_entry(void *pvParameters)
{
    FSP_PARAMETER_NOT_USED (pvParameters);

    /* TODO: add your own code here */
   fsp_err_t err;
    rm_hs300x_raw_data_t raw_data;
    demo_sequence_t sequence = DEMO_SEQUENCE_1;

    /* Open the Bus */
g_comms_i2c_bus0_quick_setup();

    /* Open HS300X */
g_hs300x_sensor0_quick_setup();

    while (1)
    {
        switch(sequence)
        {
        ...
    }
}
```

**After**

```c
void hs300x_sensor_thread_entry(void *pvParameters)
{
    FSP_PARAMETER_NOT_USED (pvParameters);

    /* TODO: add your own code here */
   fsp_err_t err;
    rm_hs300x_raw_data_t raw_data;
    demo_sequence_t sequence = DEMO_SEQUENCE_1;

    /* Open the Bus */
g_comms_i2c_bus0_quick_setup();

    /* Open HS300X */
g_hs300x_sensor0_quick_setup();

    while (1)
    {
        /* Switch to another thread */
vTaskDelay(1);

        switch(sequence)
        {
        ...
    }
}
3.2.3 Common function

3.2.3.1 RA family

Remove duplicates of I2C driver initialization function `g_comms_i2c_bus0_quick_setup()`, which is described in the c file for each sensor.

Leave this function only in the first sensor thread called first, remove them from the other threads.

Basically, the first thread created is called first, therefore remove from the file of the sensor created second or later.

3.2.3.2 RX family

Describe I2C driver initialization function `g_comms_i2c_bus0_quick_setup()` in (project name).c that contains `main_task()`. Please refer to `main_task()` of each sample project.

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 Change in processing of waiting for measurement completion (OB1203 sensor only)

Only when using OB1203 sensor, the measurement completion wait processing needs to be changed.

3.2.5.1 With IRQ signal

Add `vTaskDelay()` for switching to another thread in `ob1203_bio_irq_callback_wait()` as follows.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static void ob1203_bio_irq_callback_wait(void)</code></td>
<td><code>static void ob1203_bio_irq_callback_wait(void)</code></td>
</tr>
<tr>
<td>`{</td>
<td>`{</td>
</tr>
<tr>
<td>while (OB1203_BIO_CALLBACK_STATUS_WAIT == gs_irq_callback_status)</td>
<td>while (OB1203_BIO_CALLBACK_STATUS_WAIT == gs_irq_callback_status)</td>
</tr>
<tr>
<td>{</td>
<td>/* Switch to another thread */</td>
</tr>
<tr>
<td>// nothing</td>
<td>vTaskDelay(1);</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

Before:

```c
static void ob1203_bio_irq_callback_wait(void)
{
    while (OB1203_BIO_CALLBACK_STATUS_WAIT == gs_irq_callback_status)
    {
        // nothing
    }
}
```

After:

```c
static void ob1203_bio_irq_callback_wait(void)
{
    while (OB1203_BIO_CALLBACK_STATUS_WAIT == gs_irq_callback_status)
    {
        /* Switch to another thread */
        vTaskDelay(1);
    }
}
```
### 3.2.5.2 Without IRQ signal

Add `vTaskDelay()` for switching to another thread in `ob1203_bio_measurement_period_wait()` as follows.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>/* Get Device status */</td>
<td></td>
</tr>
<tr>
<td>counter = OB1203_BIO_MAX_COUNTS;</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>} while ((false == ob1203_bio_i2c_communication_success()) &amp;&amp;</td>
<td></td>
</tr>
<tr>
<td>(0 != counter));</td>
<td></td>
</tr>
<tr>
<td>if (0 == counter)</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>return false;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>while(false == *p_flag);</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>/* Get Device status */</td>
<td></td>
</tr>
<tr>
<td>counter = OB1203_BIO_MAX_COUNTS;</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>} while ((false == ob1203_bio_i2c_communication_success()) &amp;&amp;</td>
<td></td>
</tr>
<tr>
<td>(0 != counter));</td>
<td></td>
</tr>
<tr>
<td>if (0 == counter)</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>return false;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>/* Switch to another thread */</td>
<td></td>
</tr>
<tr>
<td>vTaskDelay(1);</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>while(false == *p_flag);</td>
<td></td>
</tr>
</tbody>
</table>
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 Add `tx_thread_sleep()`

Add `tx_thread_sleep()` for switching another thread to (sensor name)_thread_entry() of each sensor.

#### Before

```c
void hs300x_sensor_thread_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err;
    rm_hs300x_raw_data_t raw_data;
    demo_sequence_t sequence = DEMO_SEQUENCE_1;

    /* Open the Bus */
    g_comms_i2c_bus0_quick_setup();

    /* Open HS300X */
    g_hs300x_sensor0_quick_setup();

    while (1)
    {
        switch(sequence)  
            ...
    }
}
```

#### After

```c
void hs300x_sensor_thread_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err;
    rm_hs300x_raw_data_t raw_data;
    demo_sequence_t sequence = DEMO_SEQUENCE_1;

    /* Open the Bus */
    g_comms_i2c_bus0_quick_setup();

    /* Open HS300X */
    g_hs300x_sensor0_quick_setup();

    while (1)
    {
        /* Switch to another thread */
        tx_thread_sleep(1);

        switch(sequence)  
            ...
    }
}
3.3.3 Common function
Remove duplicates of I2C driver initialization function `g_comms_i2c_bus0_quick_setup()`, which is described in the c file for each sensor.

Leave this function only in the first sensor thread called first, remove them from the other threads.

Basically, the first thread created is called first, therefore remove from the file of the sensor created second or later.

```c
/* Quick setup for g_comms_i2c_bus0. */
void g_comms_i2c_bus0_quick_setup(void)
{
    /* ...

    Remove

    if (RSP_CFG_RLDS)
    {
        g_comms_i2c_bus0_extended_cfg.p_driver_instance = g_comms_i2c_bus0_extended_cfg.p_driver_instance;
    }
    / * Create a semaphore for blocking if a semaphore is not NULL */
    Semaphore * p_blocking_semaphore = NULL;
    if (p_blocking_semaphore)
    {
        / * Create a recursive mutex for bus lock if a recursive mutex is not NULL */
        Mutex * p_recursive_mutex = NULL;
        if (p_recursive_mutex)
        {
            / * ...

            Remove

            /* Sensor Thread entry function */
            void zm04400_sensor_thread_entry(void)
            {
                / * TOD0: add your own code here */
                ...
            }

            Remove

            / * ...

            Open the Bus */
            g_comms_i2c_bus0_quick_setup();
        }
        while(1)
        {
            / * ...
``` 

3.3.4 Sampling period
Depending on the sensor, it may be necessary to read from the sensor at a specific sampling period. In such cases, add `tx_thread_sleep()` for switching to another thread or changing the priority of threads in order to comply with the sampling period.
### 3.3.5 Change in processing of waiting for measurement completion (OB1203 sensor only)

Only when using OB1203 sensor, the measurement completion wait processing needs to be changed.

#### 3.3.5.1 With IRQ signal

Add `tx_thread_sleep()` for switching to another thread in `ob1203_bio_irq_callback_wait()` as follows.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>static void ob1203_bio_irq_callback_wait(void)</td>
</tr>
<tr>
<td></td>
<td>{ while (OB1203_BIO_CALLBACK_STATUS_WAIT == gs_irq_callback_status)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>static void ob1203_bio_irq_callback_wait(void)</td>
</tr>
<tr>
<td></td>
<td>{ while (OB1203_BIO_CALLBACK_STATUS_WAIT == gs_irq_callback_status)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
### 3.3.5.2 Without IRQ signal

Add `tx_thread_sleep()` for switching to another thread in `ob1203_bio_measurement_period_wait()` as follows.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>do</td>
<td>do</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>/* Get Device status */</td>
<td>/* Get Device status */</td>
</tr>
<tr>
<td>counter = OB1203_BIO_MAX_COUNTS;</td>
<td>counter = OB1203_BIO_MAX_COUNTS;</td>
</tr>
<tr>
<td>do</td>
<td>do</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>} while ((false == ob1203_bio_i2c_communication_success()) &amp;&amp; (0 != counter));</td>
<td>} while ((false == ob1203_bio_i2c_communication_success()) &amp;&amp; (0 != counter));</td>
</tr>
<tr>
<td>if (0 == counter)</td>
<td>if (0 == counter)</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>return false;</td>
<td>return false;</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>while(false == *p_flag);</td>
<td>/* Switch to another thread */</td>
</tr>
<tr>
<td></td>
<td>tx_thread_sleep(1);</td>
</tr>
<tr>
<td></td>
<td>} while(false == *p_flag);</td>
</tr>
</tbody>
</table>
4. Appendix

4.1.1 API processing period

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

<table>
<thead>
<tr>
<th>Item</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Board</td>
<td>RTK7EKA6M4S00001BE (EK-RA6M4)</td>
</tr>
<tr>
<td>MCU</td>
<td>RA6M4 (R7FA6M4AF3CFB 144pin)</td>
</tr>
<tr>
<td>Clock</td>
<td>200MHz</td>
</tr>
<tr>
<td>Voltage</td>
<td>5V</td>
</tr>
<tr>
<td>Environment</td>
<td>e² Studio 2022-07</td>
</tr>
<tr>
<td>C compiler</td>
<td>GCC 10.3.1.20210824</td>
</tr>
<tr>
<td>FSP</td>
<td>V.4.0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>API</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM_ZMOD4XXX_Open</td>
<td>233ms</td>
</tr>
<tr>
<td>RM_ZMOD4XXX_MeasurementStart</td>
<td>2.5us</td>
</tr>
<tr>
<td>RM_ZMOD4XXX_Read</td>
<td>2.5us</td>
</tr>
<tr>
<td>RM_ZMOD4XXX_Iaq2ndGenDataCalculate</td>
<td>3.2us (in stabilization)</td>
</tr>
<tr>
<td></td>
<td>637us (after stabilization is complete)</td>
</tr>
<tr>
<td>RM_ZMOD4XXX_StatusCheck</td>
<td>3us</td>
</tr>
<tr>
<td>RM_ZMOD4XXX_DeviceErrorCheck</td>
<td>2.5us</td>
</tr>
</tbody>
</table>
## Revision History

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Sep 30, 2022</td>
<td>-</td>
<td>-</td>
<td>First Release</td>
</tr>
</tbody>
</table>
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 implants; 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. 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 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
TOYSU FORESTIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

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/

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

© 2022 Renesas Electronics Corporation. All rights reserved.