RAファミリ、RXファミリ、RL78ファミリ、RZファミリ
センサソフトウェア組み合わせマニュアル

要旨

本アプリケーションノートでは、RA ファミリ、RX ファミリ、RL78 ファミリ、RZ ファミリで動作するセンサソフトウェアプロジェクトを組み合わせて使用する際のコード変更方法について説明します。

動作確認デバイス

RA6M4 グループ

RX65N グループ

RL78/G23 グループ

RL78/G14 グループ

RZ/G2L グループ

参照ドキュメント

HS300x サンプルソフトウェアマニュアル (R01AN5897)

HS400x サンプルソフトウェアマニュアル (R01AN6333)

FS2012 サンプルソフトウェアマニュアル (R01AN6047)

FS3000 サンプルソフトウェアマニュアル (R01AN5898)

FS1015 サンプルソフトウェアマニュアル (R01AN6049)

ZMOD4xxx サンプルソフトウェアマニュアル (R01AN5899)

OB1203 サンプルソフトウェアマニュアル (R01AN6311)

商標・他社 TM

FreeRTOS™ と FreeRTOS.org™ は Amazon Web Serices, Inc.の登録商標です。

Microsoft® Azure RTOS は米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

目次

1. 概要 3

2. I2Cバス 3
2.1 RA, RZ 4
2.2 RL78/G14 6
2.3 RX, RL78/G2x 9
2.4 I2Cバスのセットアップ（RA, RX, RZのみ） 12

2.4.1 NonOS 12
2.4.2 FreeRTOS/Azure 13

3. コード変更手順 16
3.1 NonOS 17
3.1.1 ファイルのコピー＆ペースト 17
3.1.2 初期化処理とメイン処理の追加 18
3.1.3 タイマモジュールの設定 20
3.1.4 タイマモジュールAPI（RX, RL78/G14, RL78/G2Xのみ） 20
3.1.5 インクルード処理の追加（RL78/G14のみ） 26
3.2 FreeRTOS 27
3.2.1 ファイルの上書き 27
3.2.2 共通関数（RA, RZのみ） 28
3.2.3 初期化処理とメイン処理の追加 （RXのみ） 29
3.2.4 サンプリング周期 29
3.2.5 センサリセット処理の有効化（RA, RZかつZMOD4xxxセンサのみ） 29
3.3 Azure 30
3.3.1 ファイルの上書き 30
3.3.2 共通関数 31
3.3.3 サンプリング周期 32
3.3.4 センサリセット処理の有効化（ZMOD4xxxセンサのみ） 32

4. プロジェクトの設定 33
4.1 RL78/G14 33
4.2 RX（ZMOD4xxxセンサのみ） 37

5. 付録 39
5.1 API処理時間 39

改訂記録 40

製品ご使用上の注意事項 41

ご注意書き 42

1. 概要

センサソフトウェアサンプルプロジェクトを組み合わせて使用するには、以下の変更が必要となります。

- I2C バスのコンフィグレーションの変更
- RTOS に応じたファイル構成およびコードの変更

2. I2C バス

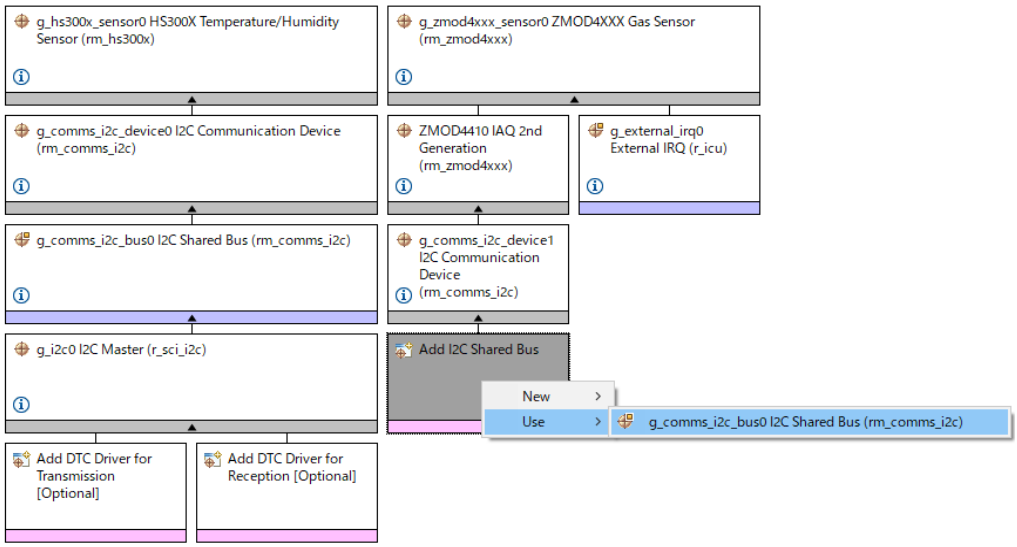
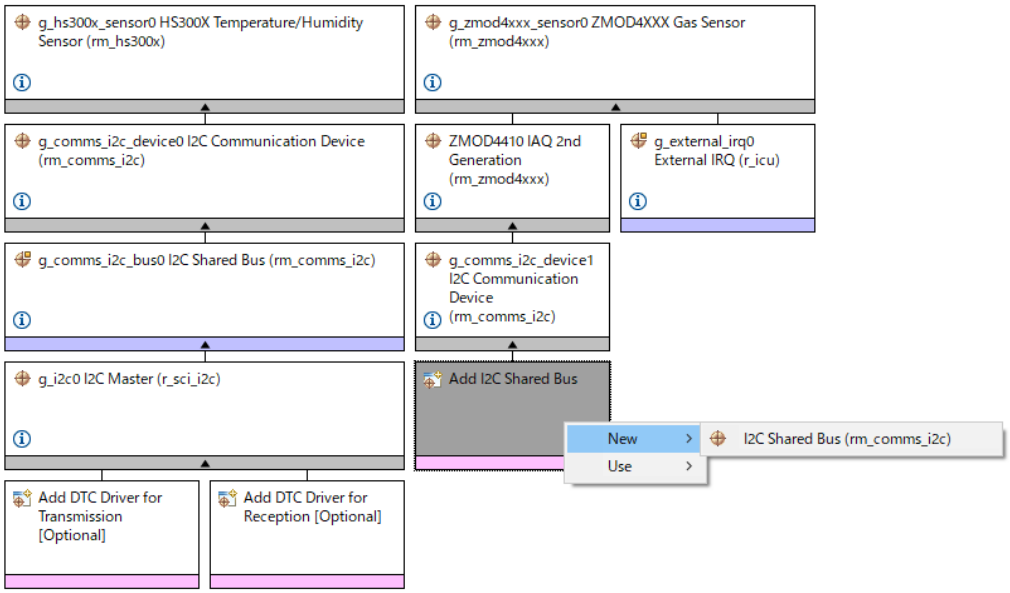
本章では、複数のセンサで同じ I2C バスを共有する場合と、異なる I2C バスを使用する場合に必要な処理を紹介します。使用する MCU でコンフィグレーション設定は異なります。

※ HS3001 と ZMOD4410 を組み合わせて使用する例で説明します。他センサの組み合わせでも同様の方法になります。

2.1　RA, RZ

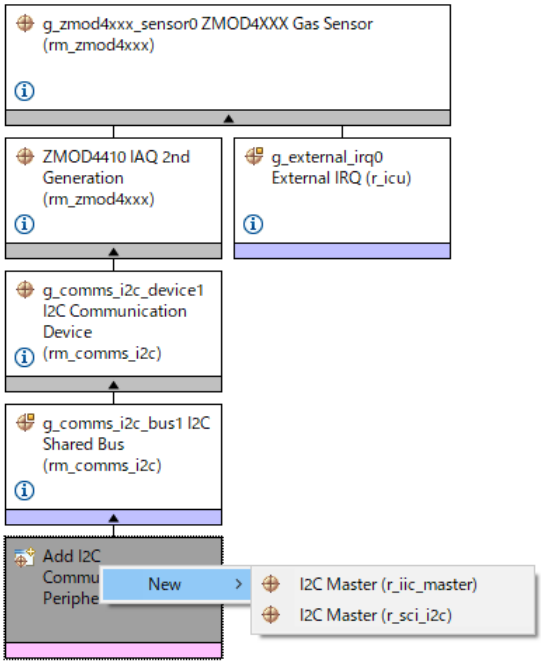
HS300x スタックを追加した状態で、ZMOD4xxx スタックを追加します。以下のように”Add I2C Shared Bus”が選択可能な状態になります。

- ”Use”を選択すると、HS3001 と ZMOD4410 で I2C バスを共有することが可能です。
- ”New”を選択すると、HS3001 と ZMOD4410 で異なる I2C バスを使用することが可能です。

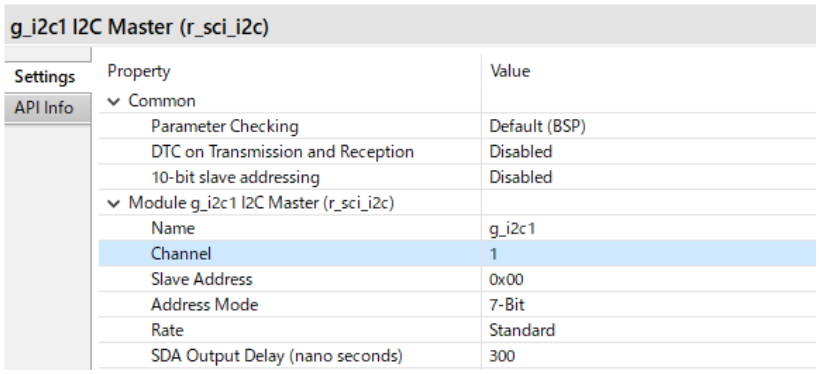
I2C バスを 共有する場合	
異なる I2C バスを 使用する場合	

”Add I2C Shared Bus”で”New”を選択すると、以下のように”Add I2C Communications Peripheral”が選択可能な状態になります。

異なるバスを使用する場合、使用する周辺機能を選択してください。

異なる I2C バスを使用する場合	

また、選択した周辺機能のプロパティから使用するチャネル番号を設定してください。

異なる I2C バスを使用する場合	

2.2　RL78/G14

以下の手順は3.1.1ファイルのコピー＆ペーストを実行した上で行ってください。

r_(sensor_name)_rl_config.h ファイルを開きます。HS3001 と ZMOD4410 の 2 つのセンサを使用するため、”RM_HS300X_CFG_DEVICE0_COMMS_INSTANCE”の値を g_comms_i2c_device0、”RM_ZMOD4XXX_CFG_DEVICE0_COMMS_INSTANCE”の値を g_comms_i2c_device1 に設定します。

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

r_comms_i2c_rl_config.h ファイルを開きます。HS3001 と ZMOD4410 の 2 つのセンサを使用するため、”COMMS_I2C_CFG_DEVICE_NUM_MAX”の値を 2 に設定します。

- I2C バスは共有する場合は、”COMMS_I2C_CFG_BUS_NUM_MAX”の値を 1 のまま変更しません。
- 異なる I2C バスを使用する場合は、”COMMS_I2C_CFG_BUS_NUM_MAX”の値を 2 に設定します。

I2C バスを 共有する場合	/* SPECIFY NUMBER OF BUSES */ #define COMMS_I2C_CFG_BUS_NUM_MAX　(1) /* SPECIFY NUMBER OF DEVICES */ #define COMMS_I2C_CFG_DEVICE_NUM_MAX　(2)
異なる I2C バスを 使用する場合	/* SPECIFY NUMBER OF BUSES */ #define COMMS_I2C_CFG_BUS_NUM_MAX　(2) /* SPECIFY NUMBER OF DEVICES */ #define COMMS_I2C_CFG_DEVICE_NUM_MAX　(2)

次に、”COMMS_I2C_CFG_DEVICE1_BUS_CH”の設定を行います。

- I2C バスを共有する場合は、同じ bus 番号の” g_comms_i2c_bus0_extended_cfg”を設定します。
- 異なる I2C バスを使用する場合は、異なる bus 番号の” g_comms_i2c_bus1_extended_cfg”を設定します。

また、”COMMS_I2C_CFG_DEVICE1_SLAVE_ADDR”の値を 0x32 に、
“COMMS_I2C_CFG_DEVICE1_CALLBACK”の値を rm_zmod4xxx_callback0 に設定します。

I2C バスを共有する場合	`/* 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 */`

異なる I2C バスを使用する場合	`/* 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 */`

次に、”COMMS_I2C_CFG_BUS1_DRIVER_TYPE”と、”COMMS_I2C_CFG_BUS1_DRIVER_CH”の設定を行います。

- I2C バスを共有する場合は、変更はありません。
- 異なる I2C バスを使用する場合は、使用する Driver Type とチャネル番号を設定します。

異なる I2C バスを使用する場合	`/* 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. */`

異なるバスを使用する場合、コールバック関数の設定を行います。

r_cg_serial_user.c ファイルを開き、使用するチャネルのコールバック関数に、rm_comms_i2c_bus1_callback()関数を追加します。

送受信完了コールバックでは、引数を false に、エラーコールバックでは、引数を true に設定してください。

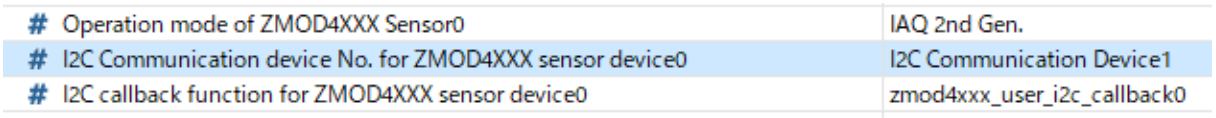
```
/***********************************************************************************************************
* Function Name: r_iic00_callback_master_error
* Description  : This function is a callback function when IIC00 master error occurs.
* Arguments    : flag -
*                    status flag
* 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 */
}
```

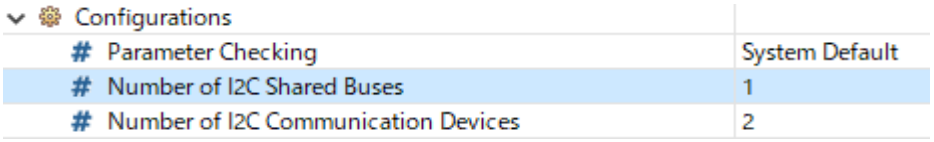
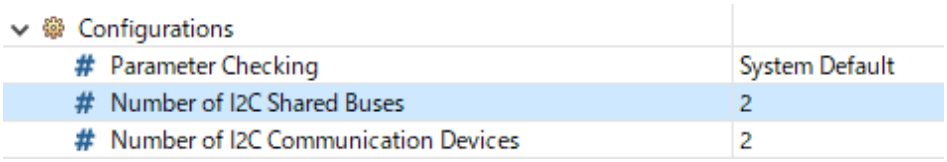
2.3　RX, RL78/G2x

r_(sensor_name)を選択します。HS3001 と ZMOD4410 の 2 つのセンサを使用するため、"I2C communication device No. for HS300x sensor device0"に I2C Communication Device0 を、"I2C communication device No. for ZMOD4XXX sensor device0"に I2C Communication Device1 を設定します。



r_comms_i2c を選択します。HS3001 と ZMOD4410 の 2 つのセンサを使用するため、”Number of I2C Communication Devices”を 2 に設定します。

- I2C バスは共有する場合は、”Number of I2C Shared Buses”は、1 のまま変更しません。
- 異なる I2C バスを使用する場合は、”Number of I2C Shared Buses”を 2 に設定します。

I2C バスを 共有する場合	
異なる I2C バスを 使用する場合	

次に、”I2C Shared Bus No. for I2C Communication Device1”の設定を行います。

- I2C バスを共有する場合は、同じ I2C Share Bus 番号の”I2C Shared Bus0”を設定します。
- 異なる I2C バスを使用する場合は、異なる I2C Shared Bus 番号の”I2C Shared Bus1”を設定します。

また、”Slave address for I2C Communication Device1”の値を 0x32 に
“Callback function for I2C Communication Device1”の値を rm_zmod4xxx_callback0 に設定します。

I2C バスを 共有する場合	# I2C Shared Bus No. for I2C Communication Device0 : I2C Shared Bus0 # Slave address for I2C Communication Device0 : 0x44 # Address mode for I2C Communication Device0 : 7bit address mode # Callback function for I2C Communication Device0 : rm_hs300x_callback0 # Timeout for the blocking bus of I2C Communication Device : 0xFFFFFFFF # I2C Shared Bus No. for I2C Communication Device1 : I2C Shared Bus0 # Slave address for I2C Communication Device1 : 0x32 # Address mode for I2C Communication Device1 : 7bit address mode # Callback function for I2C Communication Device1 : rm_zmod4xxx_callback0 # Timeout for the blocking bus of I2C Communication Device : 0xFFFFFFFF
異なる I2C バスを 使用する場合	# I2C Shared Bus No. for I2C Communication Device0 : I2C Shared Bus0 # Slave address for I2C Communication Device0 : 0x44 # Address mode for I2C Communication Device0 : 7bit address mode # Callback function for I2C Communication Device0 : rm_hs300x_callback0 # Timeout for the blocking bus of I2C Communication Device : 0xFFFFFFFF # I2C Shared Bus No. for I2C Communication Device1 : I2C Shared Bus1 # Slave address for I2C Communication Device1 : 0x32 # Address mode for I2C Communication Device1 : 7bit address mode # Callback function for I2C Communication Device1 : rm_zmod4xxx_callback0 # Timeout for the blocking bus of I2C Communication Device : 0xFFFFFFFF

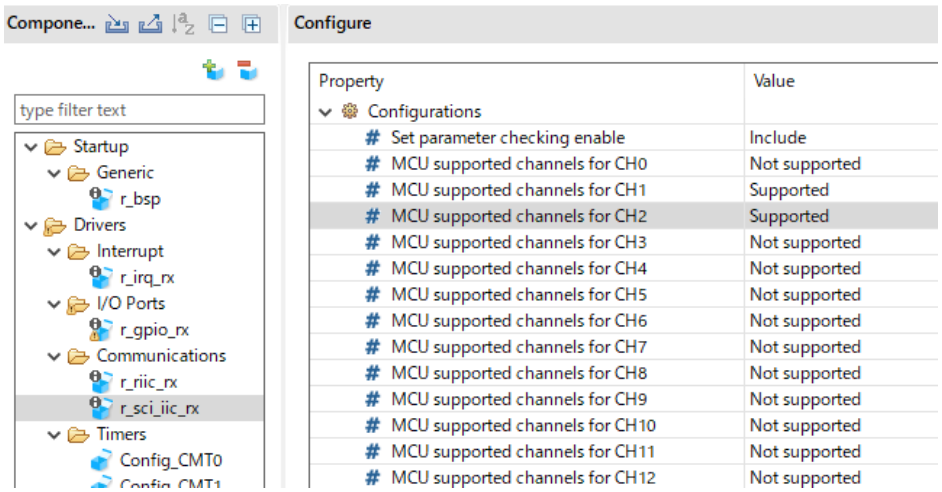
次に、”I2C Driver Type for I2C Shared Bus1”と”Channel No. for I2C Shared Bus1”の設定を行います。

- I2C バスを共有する場合は、変更はありません。
- 異なる I2C バスを使用する場合は、使用する Driver Type とチャネル番号を設定します。

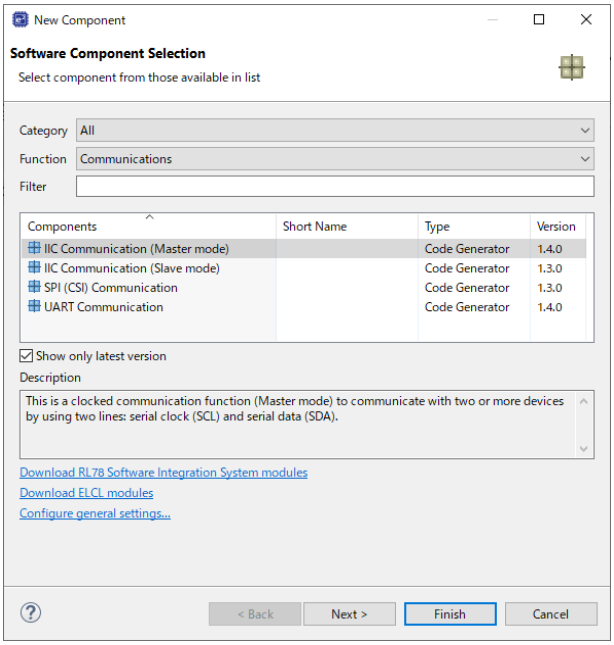
異なる I2C バスを 使用する場合	# I2C Driver Type for I2C Shared Bus0 : SCI IIC # Channel No. for I2C Shared Bus0 : 2 # Timeout for the bus lock of I2C Shared Bus0 : 0xFFFFFFFF # I2C Driver Type for I2C Shared Bus1 : SCI IIC # Channel No. for I2C Shared Bus1 : 0 # Timeout for the bus lock of I2C Shared Bus1 : 0xFFFFFFFF

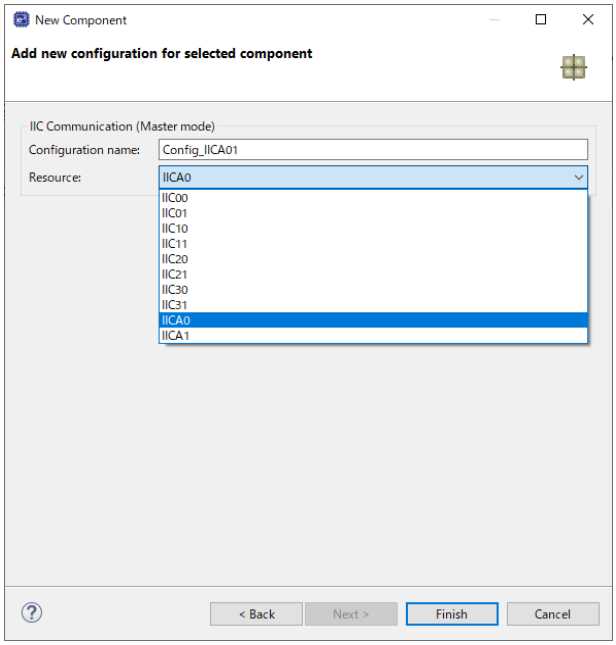
異なるバスを使用する場合、コンポーネントの追加、設定の変更を行います。

RX 向けプロジェクトの場合、I2C Shared Bus1 で使用する Driver Type とチャネル番号に応じて、r_sci_iic_rx または r_riic_rx を選択し、チャネルの有効化を行ってください。



RL78/G2x 向けプロジェクトの場合、ソフトウェアコンポーネントの追加から、I2C Shared Bus1 で使用する Driver Type とチャネル番号に応じたコンポーネントを選択してください。





2.4　I2C バスのセットアップ（RA, RX, RZ のみ）

異なるバスを使用する場合、I2C ドライバの初期化処理を追加する必要があります。

※　テキストボックス内のコード例には RA 向けプロジェクトのコードを使用しています。他 MCU 向けプロジェクトでも同様の方法になります。

RL78 MCU の場合、Code Generator や Smart Configurator が初期化処理を生成するため、初期化処理を追加する必要はありません。

2.4.1　NonOS

main()または hal_entry()が宣言されている C ファイルで、g_comms_i2c_bus0_quick_setup()をコピー＆ペーストします。

関数の名称を g_comms_i2c_bus1_quick_setup()に、関数内で参照する I2C インスタンスを g_comms_i2c_bus1_extended_cfg に変更します。

```
/* 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();
    }
}
```

g_comms_i2c_bus1_quick_setup()を main()または hal_entry()で呼び出してください。

2.4.2　FreeRTOS/Azure

※　RX 向けかつ FreeRTOS を使用するプロジェクトの場合、2.4.1 NonOSを参照してください。

関数定義の例は FreeRTOS, Azure に分けて記載します。

sensor_thread_common.c ファイルで、g_comms_i2c_bus0_quick_setup()をコピー＆ペーストします。

関数の名称を g_comms_i2c_bus1_quick_setup()に、関数内で参照する I2C インスタンスを g_comms_i2c_bus1_extended_cfg に変更します。

また、関数内で設定する I2C ドライバ初期化完了フラグを g_comms_i2c_bus1_setup に変更します。

- FreeRTOS の場合

```
/* 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 の場合

```
/* 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;
}
```

次に、I2C ドライバ初期化完了フラグを定義してください。

```
bool g_comms_i2c_bus1_setup = false;
```

sensor_thread_common.h に以下のプロトタイプ宣言、extern 宣言を追加してください。

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

I2C Shared Bus1 を使用するセンサタスクもしくは、スレッドの(sensor_name)_sensor_thread_entry()にて、g_comms_i2c_bus0_quick_setup()の呼出処理を以下のように変更してください。

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

3.　コード変更手順

本章では、I2C 通信完了時または IRQ 信号待機時にセンサを切り替えながら動作するためのコード変更手順を説明します。

コード変更の前に、各サンプルプロジェクトを参考に、Smart Configurator や Code Generator 上の設定を行う必要があります。（RL78, RX MCU の場合、生成されたドライバコードにコールバック等の関数を追加する必要があります。）

※　MCU は RA、センサは HS3001 と ZMOD4410 を使用する例で説明しています。MCU やセンサの種別により異なる変更が必要な場合は説明を追加しています。

RL78/G2x で BSP v1.30 以前のバージョンを使用する場合、多重定義を防ぐために以下の変更が必要です。

各センサモジュール内の rm_(sensor name)_common.c に定義されている変数 bsp_delay_time を以下のように変更します。変更後、各センサモジュール内の rm_(sensor name)_common_(compiler name).asm ファイルを削除します。

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

変更後	`extern const unsigned long long bsp_delay_time[];`

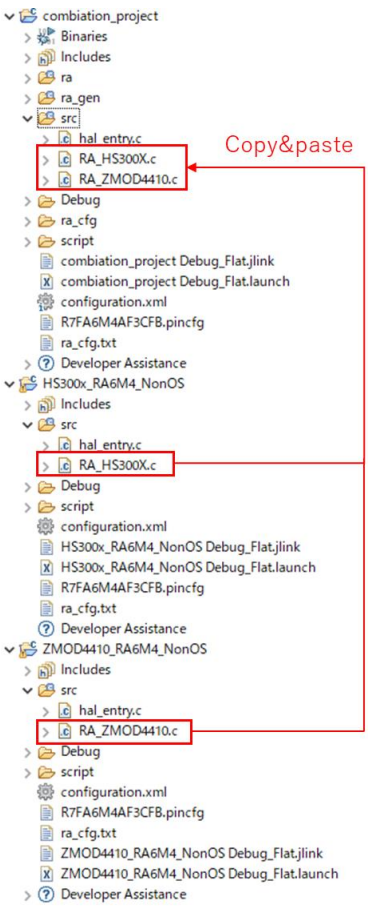
コードの変更手順を NonOS, FreeRTOS, Azure に分けて説明します。

3.1　NonOS

※　ZMOD4410 センサ向けの NonOS プロジェクトの v1.52-v1.53 を使用して他センサとの組み合わせを行う場合は、使用するサンプルプログラムの最新のバージョンへ切り替えをお願いします。

3.1.1　ファイルのコピー＆ペースト

各サンプルプロジェクトからアプリケーションが記述されている C ファイルをコピー＆ペーストします。



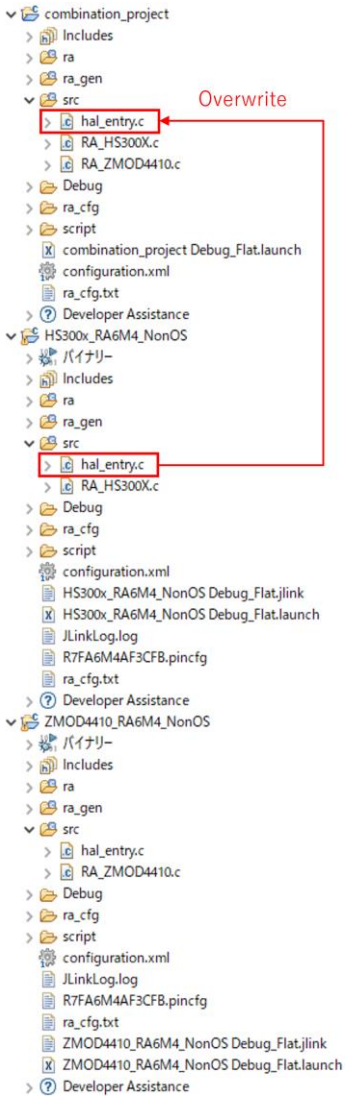
OB1203 センサの場合、さらに ob1203_bio フォルダもコピー＆ペーストする必要があります。

RL78/G14 向けプロジェクトの場合、各センサソフトウェアプロジェクトから application フォルダ、general フォルダ、r_bsp フォルダ、r_comms_i2c_rl フォルダ、r_config フォルダ、r_(sensor_name)フォルダをコピー＆ペーストする必要があります。

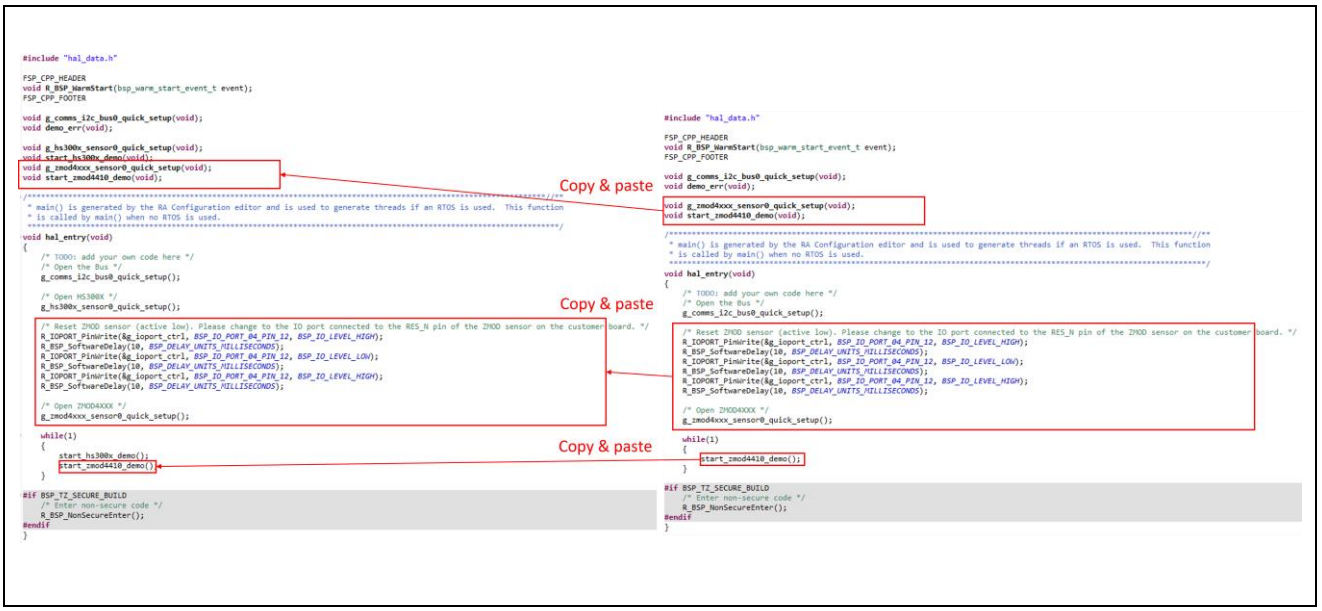
同名フォルダ及びファイルが存在する場合は、フォルダ及びファイルの上書きを行ってください。

3.1.2　初期化処理とメイン処理の追加

サンプルプロジェクトから main()または hal_entry()が宣言されている C ファイルを上書きします。



また、上記の処理にて上書きに使用しなかったサンプルプロジェクトの main()または hal_entry()が宣言されている C ファイルから、センサ初期化処理、メイン処理、プロトタイプ宣言をコピー＆ペーストします。



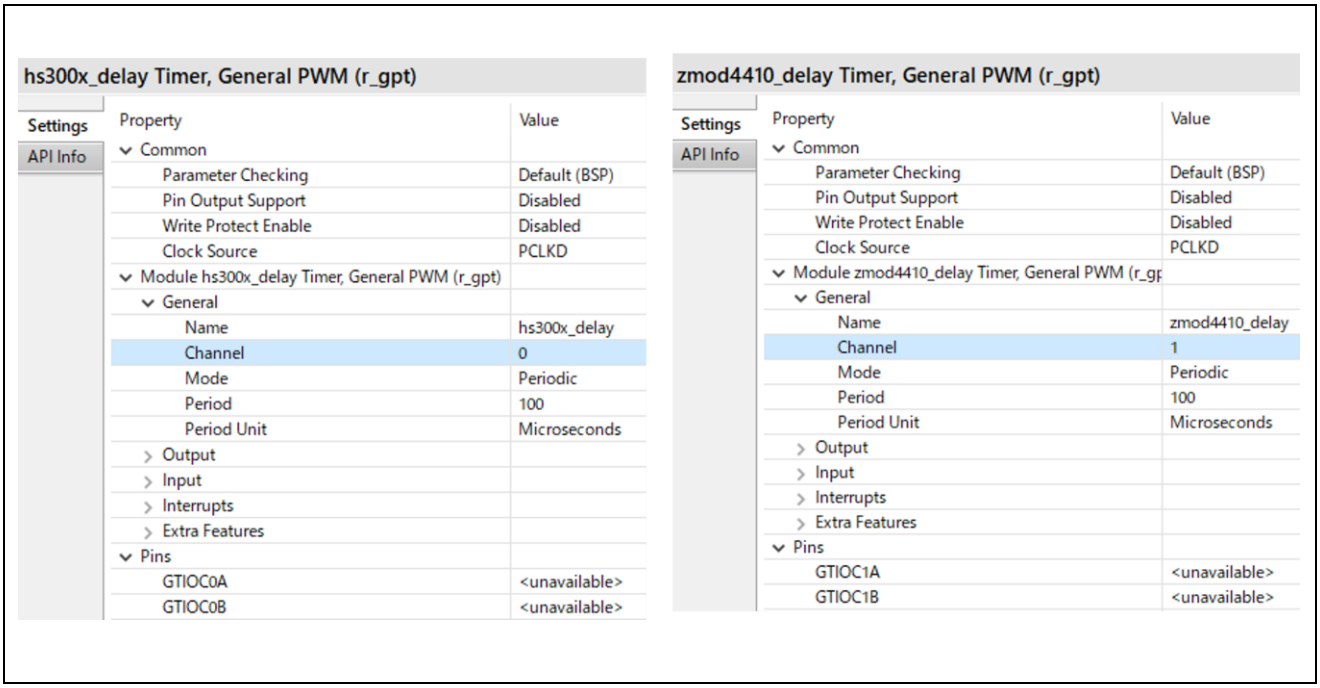
RX 向けかつ ZMOD4xxx センサの場合、r_gpio_rx_if.h のインクルード処理を追加する必要があります。

3.1.3　タイマモジュールの設定

各アプリコードで使用するタイマモジュールのチャネルを別個に設定します。

3.1.3.1　RA, RZ

FPS Configurator から以下のように設定します。



3.1.3.2　RL78G14

Code Generator にて、周辺機能から異なるチャネルのタイマを設定します。

3.1.3.3　RX, RL78/G2X

Smart Configurator にて、ソフトウェアコンポーネントの追加から、異なるチャネルのタイマコンポーネントを設定します。

3.1.4　タイマモジュール API（RX, RL78/G14, RL78/G2X のみ）

アプリコード内のタイマモジュール API を設定したチャネルと対応するものに変更します。

3.1.4.1　RX

変更前

```
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;
}
```

変更後

```
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;
}
```

また、RX 向けサンプルプロジェクトで使用されている R_Config_CMT0_Reset()はユーザ定義の関数のため、組み合わせ時にはタイマモジュールの c ファイルに新たに定義する必要があります。

c ファイルのユーザコード記述部分に以下の通り関数を定義してください。

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

加えて、タイマモジュールの h ファイルにおけるユーザコード記述部分に、R_Config_CMT0_Reset()のプロトタイプ宣言を追加してください。

3.1.4.2　RL78/G14

変更前

```
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;
}
```

変更後

```
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;
}
```

また、RL78G14 向けサンプルプロジェクトで使用されている R_TAU0_Channel0_Reset()はユーザ定義の関数のため、組み合わせ時にはタイマモジュールの c ファイルに新たに定義する必要があります。

c ファイルのユーザコード記述部分に以下の通り関数を定義してください。

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

加えて、タイマモジュールの h ファイルにおけるユーザコード記述部分に、R_TAU0_Channel0_Reset()のプロトタイプ宣言を追加してください。

3.1.4.3　RL78/G23

変更前

```
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;
}
```

変更後

```
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;
}
```

また、RL78G23 向けサンプルプロジェクトで使用されている R_Config_TAU0_0_Reset()はユーザ定義の関数のため、組み合わせ時にはタイマモジュールの c ファイルに新たに定義する必要があります。

c ファイルのユーザコード記述部分に以下の通り関数を定義してください。

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

加えて、タイマモジュールの h ファイルにおけるユーザコード記述部分に、R_Config_TAU0_0_Reset()のプロトタイプ宣言を追加してください。

3.1.5　インクルード処理の追加（RL78/G14 のみ）

ZMOD4xxx センサを使用かつ、IRQ を使用する場合、plathome.h に以下のインクルード処理が必要です。

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

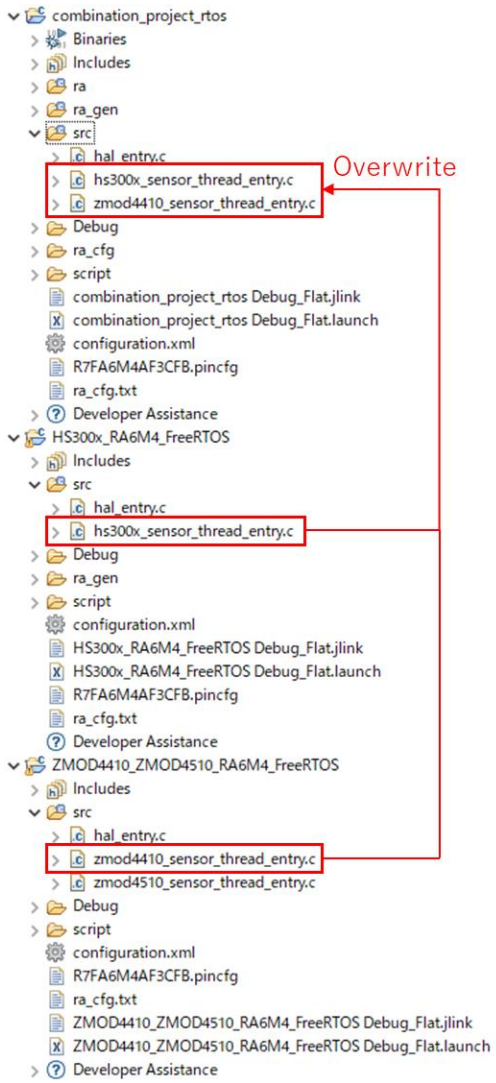
OB1203 センサを使用する場合、r_smc_entry.h に以下のインクルード処理が必要です。

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

3.2　FreeRTOS

3.2.1　ファイルの上書き

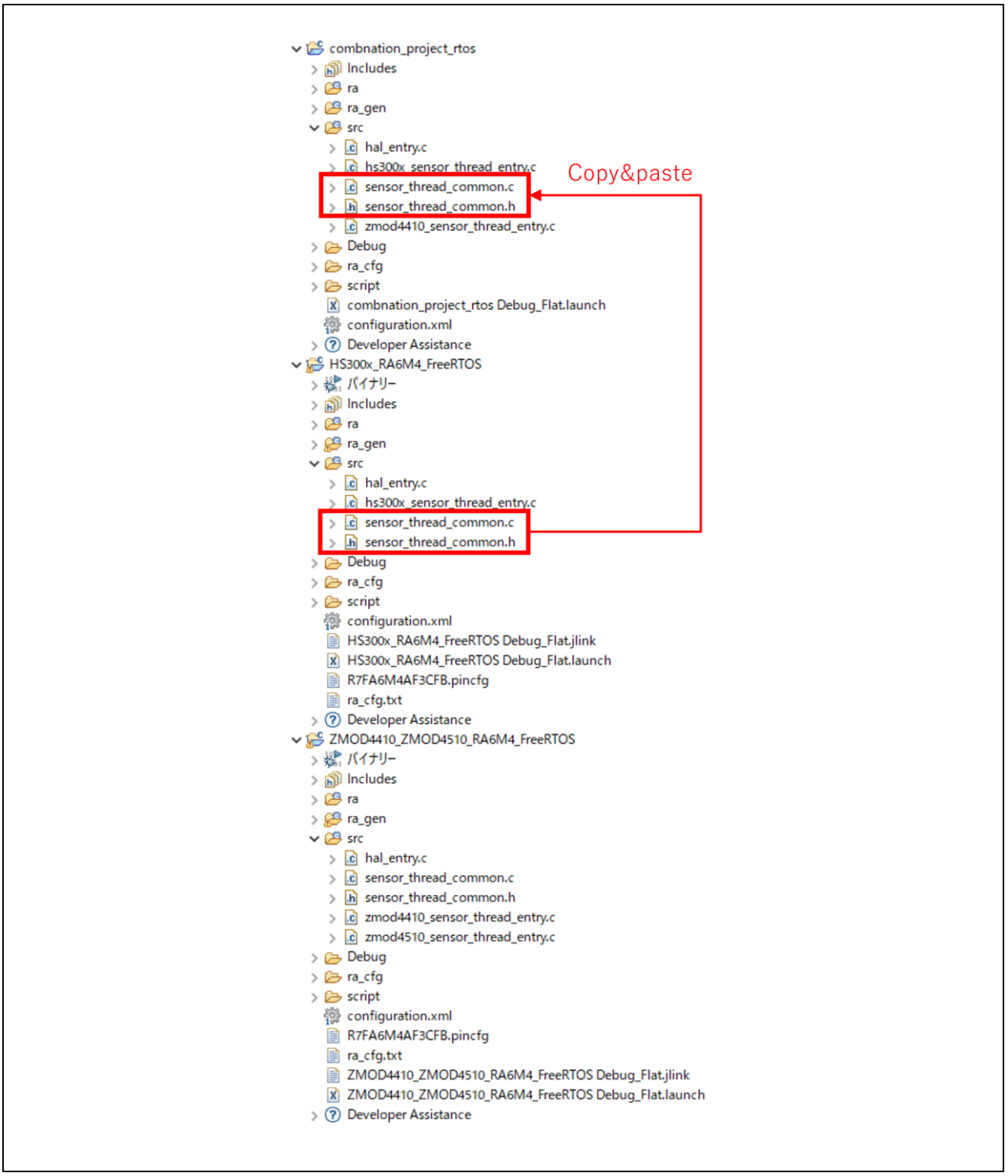
各サンプルプロジェクトからアプリケーションが記述されている C ファイルを上書きします。



OB1203 センサの場合、さらに ob1203_bio フォルダをコピー＆ペーストする必要があります。

3.2.2　共通関数（RA, RZ のみ）

サンプルプロジェクトから、共通関数を記述している sensor_thread_common.c ファイル、sensor_thread_common.h ファイルをコピー＆ペーストします。



3.2.3　初期化処理とメイン処理の追加　（RX のみ）

RX 向けかつ FreeRTOS を使用するプロジェクトの場合、3.1.2 初期化処理とメイン処理の追加を参照してください。

3.2.4　サンプリング周期

センサによって、特定のサンプリング周期でセンサからリードする必要があります。その場合はサンプリング周期を順守するため、スレッドの優先順位を変更または他センサのコードにスレッドを切り替えるための vTaskDelay()を追加してください。

3.2.5　センサリセット処理の有効化（RA, RZ かつ ZMOD4xxx センサのみ）

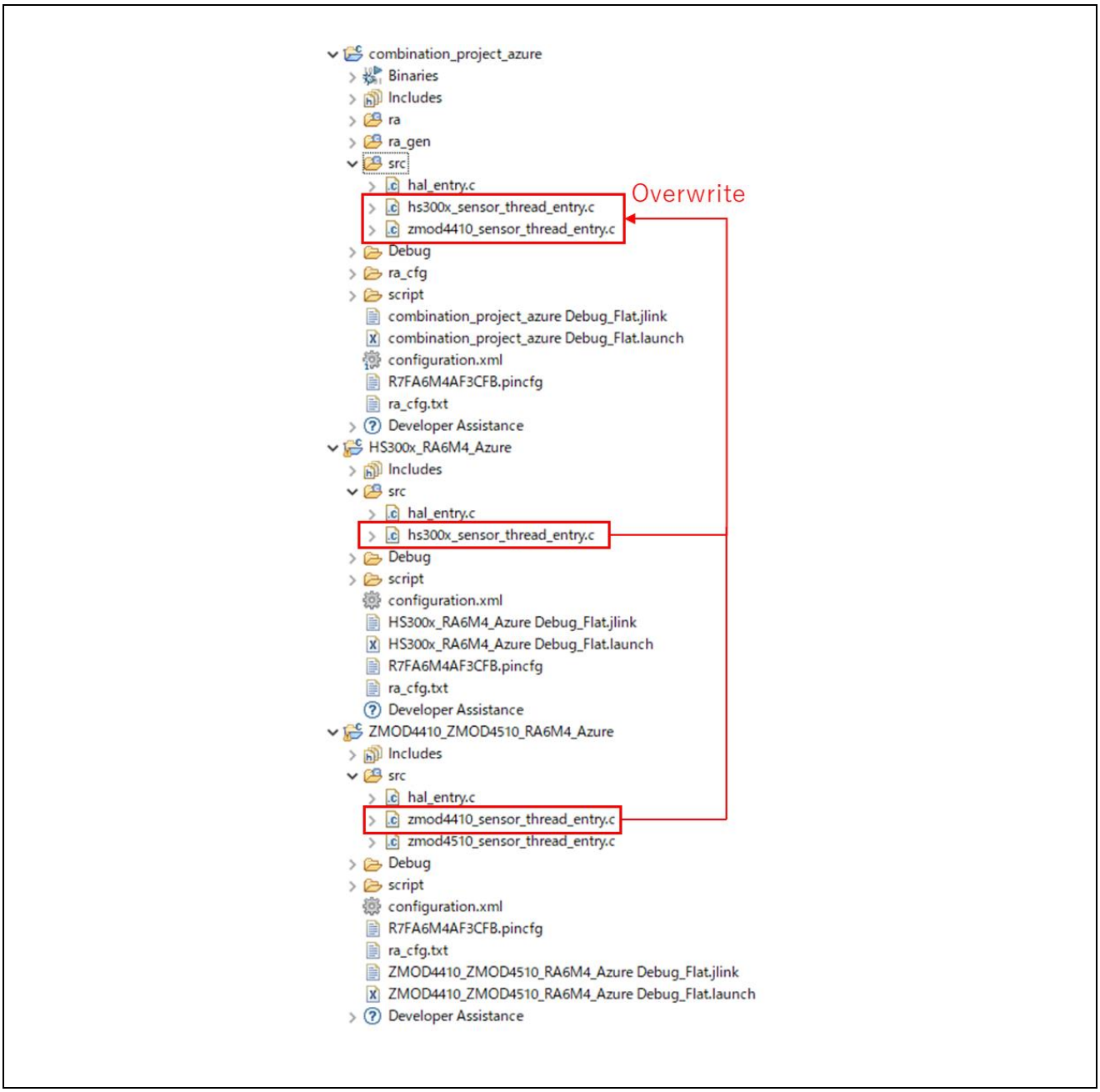
ZMOD4xxx センサを組み合わせに使用する場合、”G_ZMOD4XXX_SENSOR_RESET_ENABLE”の値を 1 に設定します。

ただし、複数の ZMOD4xxx センサを組み合わせに使用する場合、一番先に呼ばれるセンサのスレッドで定義されている”G_ZMOD4XXX_SENSOR_RESET_ENABLE”の値を 1 に設定し、それ以外のスレッドでは 0 に設定します。基本的に、最初に作成したスレッドが一番先に呼ばれます。

3.3　Azure

3.3.1　ファイルの上書き

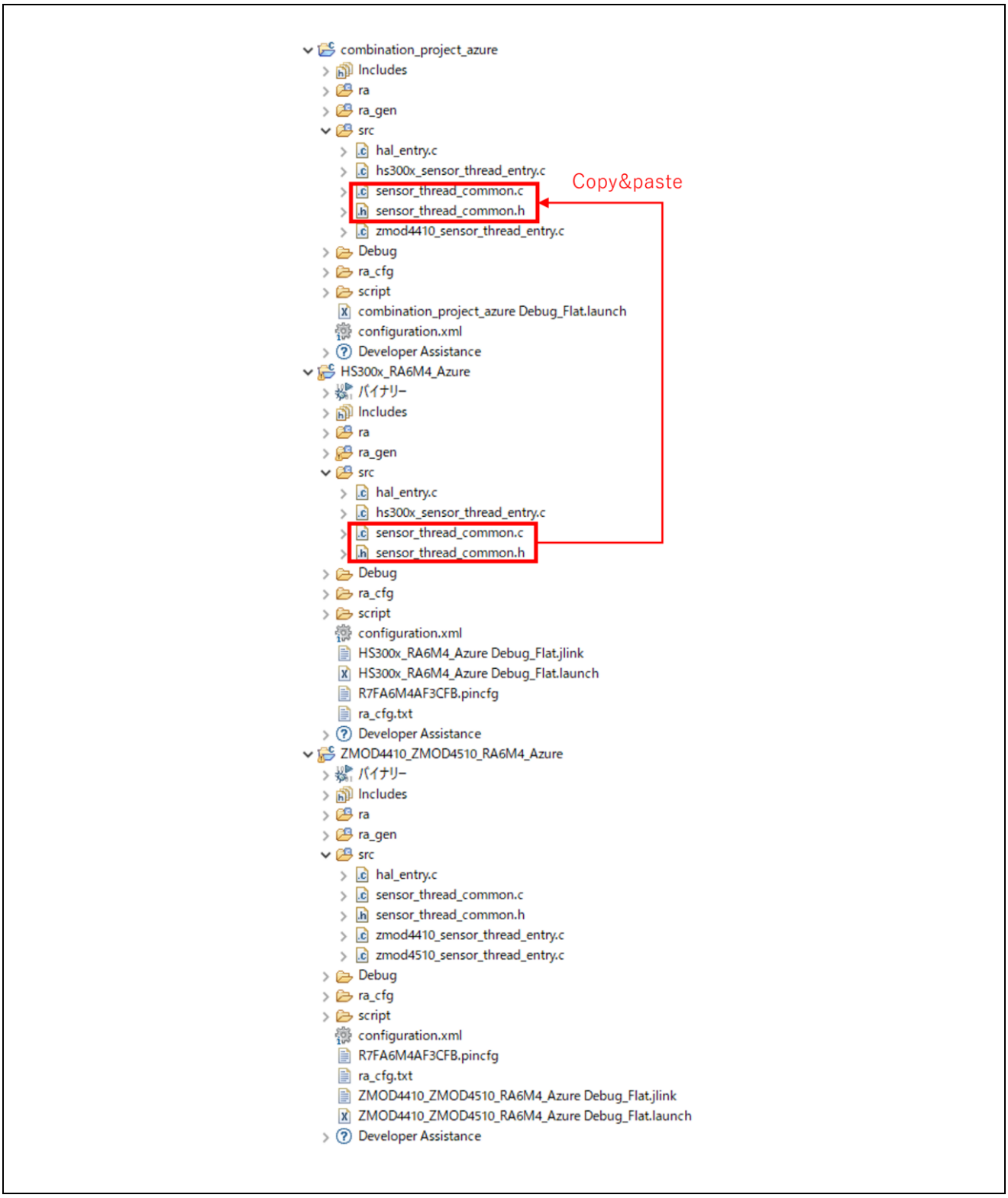
各サンプルプロジェクトからアプリケーションが記述されている C ファイルを上書きします。



OB1203 センサの場合、さらに ob1203_bio フォルダをコピー＆ペーストする必要があります。

3.3.2　共通関数

サンプルプロジェクトから、共通関数を記述している sensor_thread_common.c ファイル、sensor_thread_common.h ファイルをコピー＆ペーストします。



3.3.3　サンプリング周期

センサによって、特定のサンプリング周期でセンサからリードする必要があります。その場合はサンプリング周期を順守するため、スレッドの優先順位を変更または他センサのコードにスレッドを切り替えるための tx_thread_sleep()を追加してください。

3.3.4　センサリセット処理の有効化（ZMOD4xxx センサのみ）

ZMOD4xxx センサを組み合わせに使用する場合、”G_ZMOD4XXX_SENSOR_RESET_ENABLE”の値を 1 に設定します。

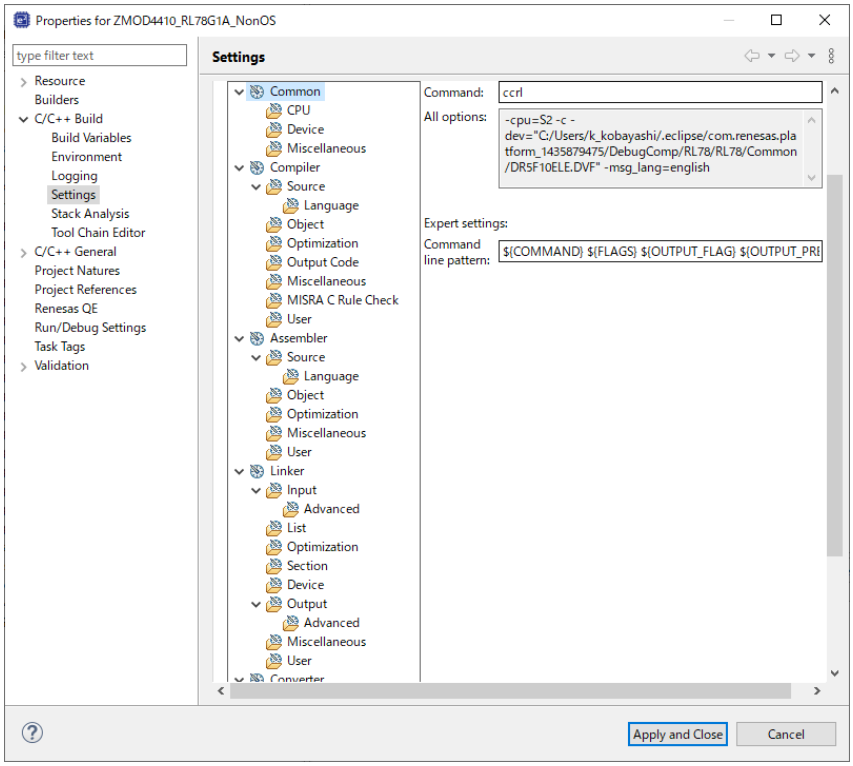
ただし、複数の ZMOD4xxx センサを組み合わせに使用する場合、一番先に呼ばれるセンサのスレッドで定義されている”G_ZMOD4XXX_SENSOR_RESET_ENABLE”の値を 1 に設定し、それ以外のスレッドでは 0 に設定します。基本的に、最初に作成したスレッドが一番先に呼ばれます。

4.　プロジェクトの設定

4.1　RL78/G14

プロジェクトのプロパティを開きます。

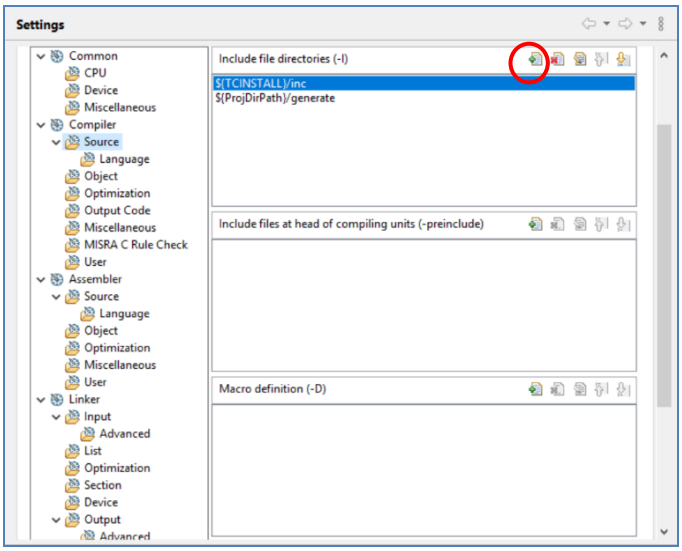
プロパティの[C/C++ Build]-[Settings]を選択し、settings を開きます。

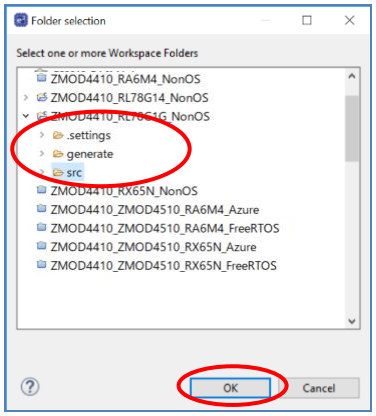


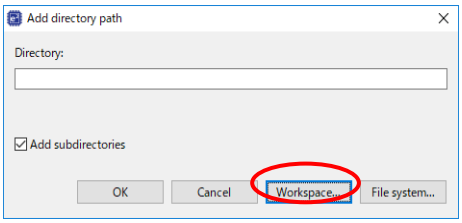
Tool Settings タブの[Compiler]-[Source]を選択し、[Add]アイコンを押下します。

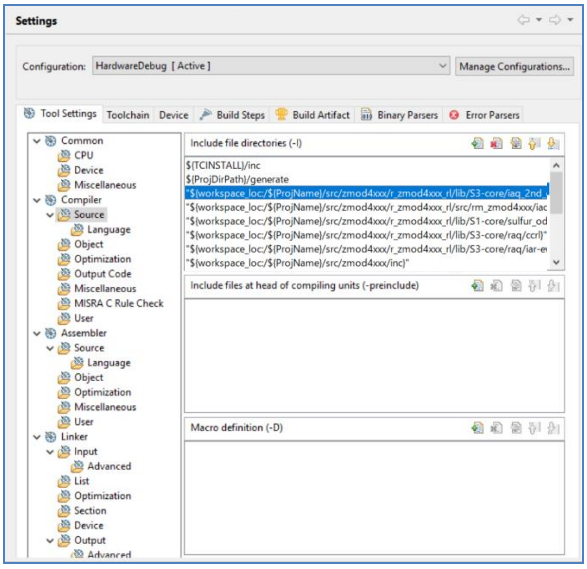
[Add directory path]ダイアログで、[Workspace]ボタンを押下し、表示されたプロジェクトの一覧から、新しく作成したプロジェクトの”src”フォルダを選択し、[OK]ボタンを押下します。

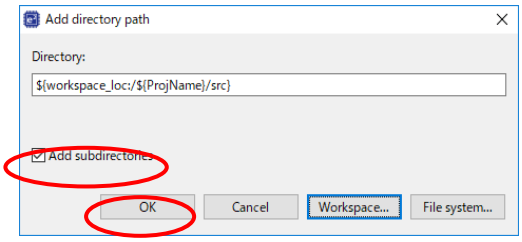
”Add subdirectroies”にチェックを入れて、[OK]ボタンを押下します。





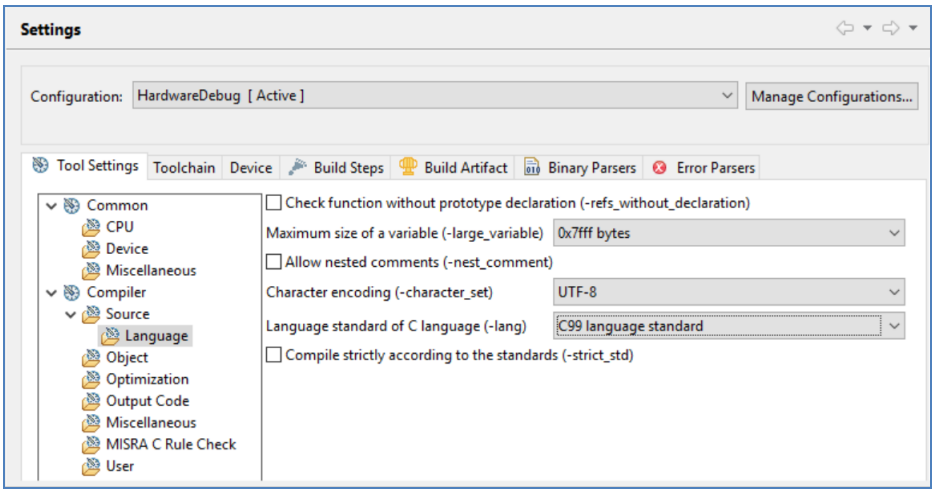






Tool settings タブの[Compiler]-[Source]-[Language]を選択し、Language standard of C language を”C99 language standard”に変更します。

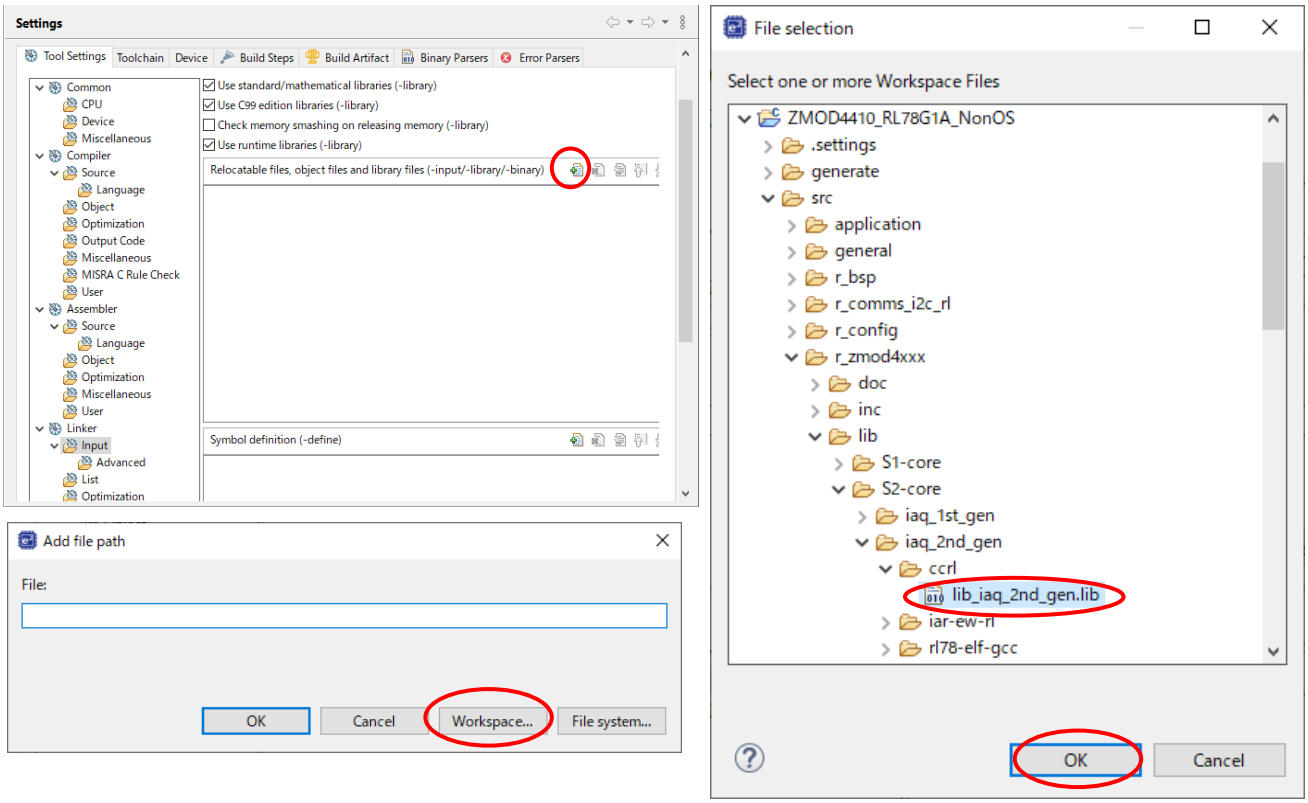
[Apply and Close]ボタンを押下して、プロパティを閉じます。

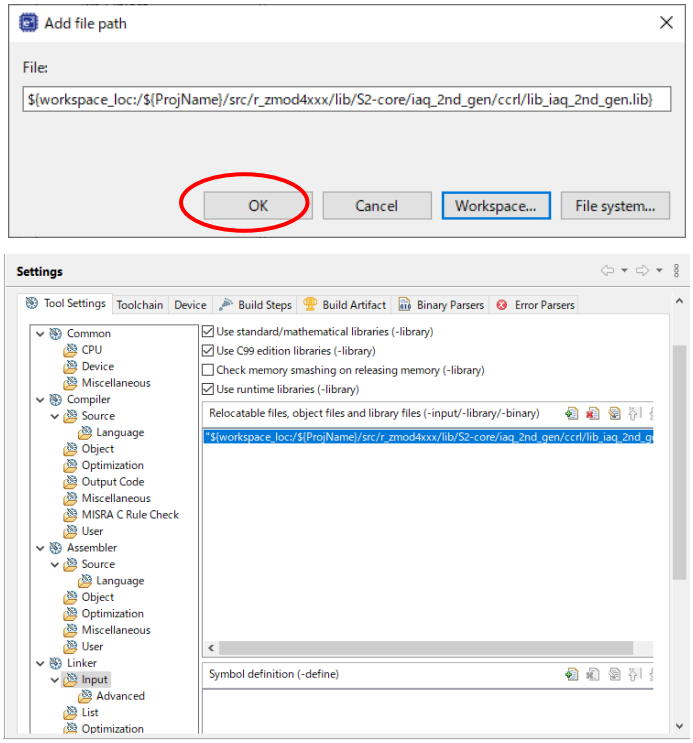


ZMOD4xxx センサを使用する場合、以下の通り[Linker]の設定の変更が必要です。

Tool settings タブの[Linker]-[Input]を選択し、[Add]アイコンを押下します。

[Add directory path]ダイアログで、[Workspace]ボタンを押下し、表示されたプロジェクトの一覧から、新しく作成したプロジェクトの”src”フォルダを選択し、”r_zmod4xxx/lib”フォルダ内から、使用する MCU アーキテクチャ、測定モード、コンパイラに応じた lib ファイルを選択してください。

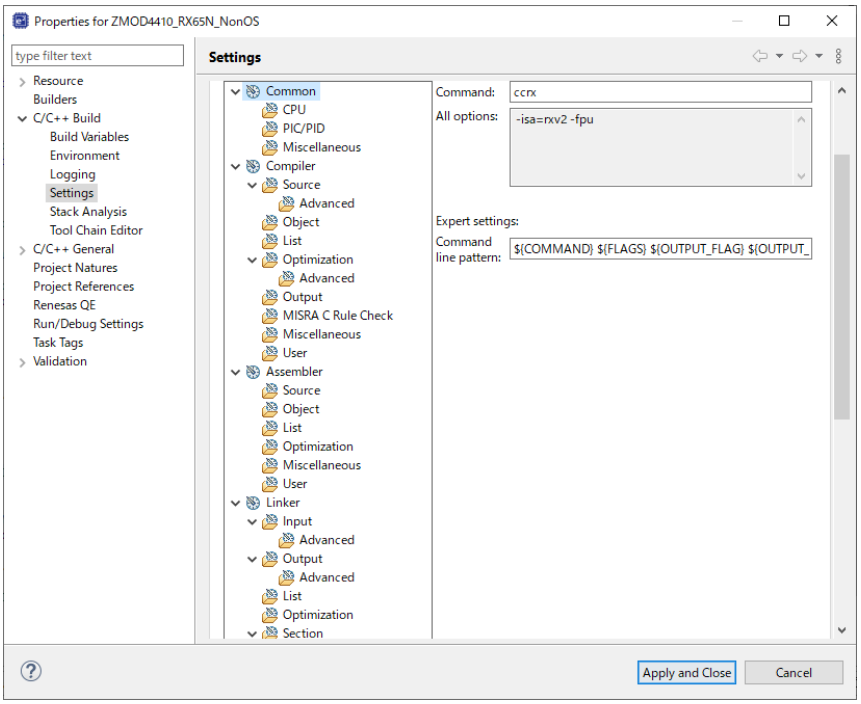




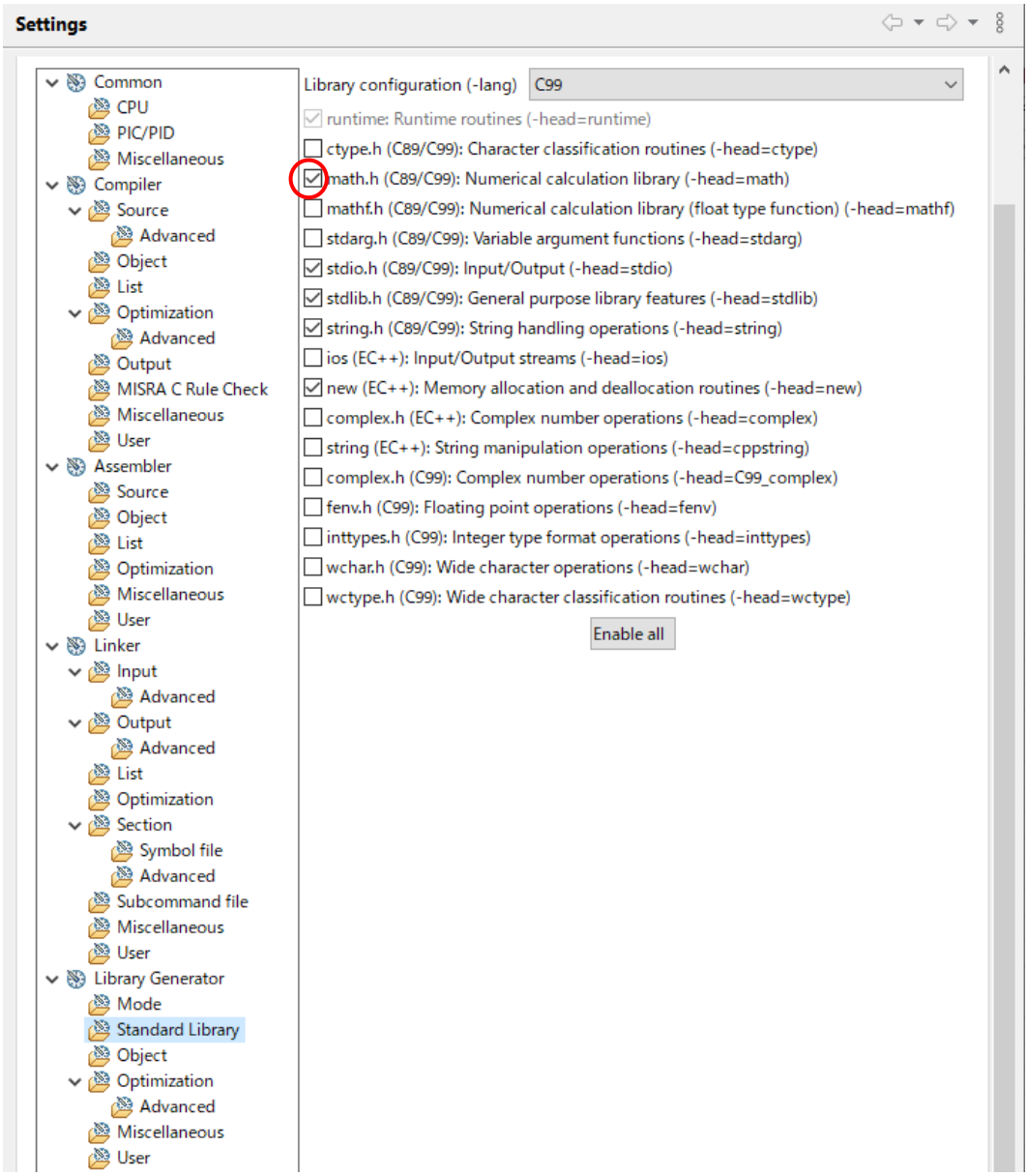
4.2　RX（ZMOD4xxx センサのみ）

プロジェクトのプロパティを開きます。

プロパティの[C/C++ Build]-[Settings]を選択し、settings を開きます。



Tool settings タブの[Library Generator]-[Standard Library]を選択し、"math.h (C89/99)”を有効にします。



5. 付録

5.1 API 処理時間

API の処理時間を説明します。ZMOD4410(IAQ 2nd Gen.) の例です。

項目	内容
デモボード	RTK7EKA6M4S00001BE (EK-RA6M4)
使用マイコン	RA6M4 (R7FA6M4AF3CFB :144pin)
動作周波数	200MHz
動作電圧	5V
統合開発環境	e2 Studio 2022-07
C コンパイラ	GCC 10.3.1.20210824
FSP	V.4.0.0

API 名	処理時間
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

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
Rev.1.00	September 30, 2022	ー	初版リリース
Rev.1.10	December 27, 2022	p5	RL78/G14 についての解説を追加
Rev.1.20	September 7, 2023	ー	I2C バスについての記載を追加 コード変更手順についての記載を削除、追加、更新

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。