

## 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<sup>™</sup> and FreeRTOS.org<sup>™</sup> 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
2.	I2C Shared Bus
2.1	RA, RZ4
2.2	RL78/G146
2.3	RX, RL78/G2x9
2.4	I2C bus setup (RA, RX, RZ Only)12
2.4.1	NonOS12
2.4.2	FreeRTOS/Azure



3.	Code change procedure	.16
3.1	NonOS	. 17
3.1.1	Copy and paste files	. 17
3.1.2	Add initialization process and main process	. 18
3.1.3	Timer module settings	. 20
3.1.4	Timer module API (RX, RL78/G14, RL78/G2X Only)	. 20
3.1.5	Add definition of including	. 26
3.2	FreeRTOS	27
3.2.1	Overwrite files	27
3.2.2	Common function (RA, RZ Only)	28
3.2.3	Add initialization process and main process (RX Only)	29
3.2.4	Sampling period	29
3.2.5	Enable sensor reset processing (RA, RZ, ZMOD4xxx sensor only)	. 29
3.3	Azure	. 30
3.3.1	Overwrite files	30
3.3.2	Common function	31
3.3.3	Sampling period	32
3.3.4	Enable sensor reset processing (ZMOD4xxx sensor only)	. 32
4.	Project settings	.33
4.1	RL78/G14	. 33
4.2	RX (ZMOD4xxx sensor only)	. 37
5.	Appendix	.39
5.1	API processing period	39
Rev	ision History	.40
Gen	eral Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products.	.41
Noti	ce	.42
Corp	porate Headquarters	.42
Con	tact information	.42
Trac	lemarks	.42



## 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





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.



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

	g_i2c1 l20	C Master (r_sci_i2c)	
The different I2C bus is used	Settings API Info	Property Common Parameter Checking DTC on Transmission and Reception 10-bit slave addressing Module a i2c1 I2C Master (r sci i2c)	Value Default (BSP) Disabled Disabled
is used		Vane Channel Slave Address	g_i2c1 1
		Address Mode Rate SDA Output Delay (nano seconds)	7-Bit Standard 300
			500



## 2.2 RL78/G14

Before applying the changes in this chapter, perform <u>3.1.1 Copy and paste files</u>.

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	*/
<pre>#define RM_ZMOD4XXX_CFG_DEVICE0_COMMS_INSTANCE</pre>	(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	<pre>/* SPECIFY NUMBER OF BUSES */ #define COMMS_I2C_CFG_BUS_NUM_MAX /* SPECIFY NUMBER OF DEVICES */ #define COMMS_I2C_CFG_DEVICE_NUM_MAX</pre>	(1) (2)
The different I2C bus is used	<pre>/* SPECIFY NUMBER OF BUSES */ #define COMMS_I2C_CFG_BUS_NUM_MAX /* SPECIFY NUMBER OF DEVICES */ #define COMMS_I2C_CFG_DEVICE_NUM_MAX</pre>	(2) (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	<pre>/* For Device No.0 */ #define COMMS_I2C_CFG_DEVICE0_BUS_CH #define COMMS_I2C_CFG_DEVICE0_SLAVE_ADDR #define COMMS_I2C_CFG_DEVICE0_CALLBACK /* For Device No.1 */ #define COMMS_I2C_CFG_DEVICE1_BUS_CH #define COMMS_I2C_CFG_DEVICE1_SLAVE_ADDR #define COMMS_I2C_CFG_DEVICE1_CALLBACK</pre>	<pre>(g_comms_i2c_bus0_extended_cfg) (0x44) /* Slave address */ (rm_hs300x_callback0) /* Callback function */ (g_comms_i2c_bus0_extended_cfg) (0x32) /* Slave address */ (rm_zmod4xxx_callback0) /* Callback function */</pre>
The different I2C bus is used	<pre>/* For Device No.0 */ #define COMMS_I2C_CFG_DEVICE0_BUS_CH #define COMMS_I2C_CFG_DEVICE0_SLAVE_ADDR #define COMMS_I2C_CFG_DEVICE0_CALLBACK /* For Device No.1 */ #define COMMS_I2C_CFG_DEVICE1_BUS_CH #define COMMS_I2C_CFG_DEVICE1_SLAVE_ADDR #define COMMS_I2C_CFG_DEVICE1_CALLBACK</pre>	<pre>(g_comms_i2c_bus0_extended_cfg) (0x44) /* Slave address */ (rm_hs300x_callback0) /* Callback function */ (g_comms_i2c_bus1_extended_cfg) (0x32) /* Slave address */ (rm_zmod4xxx_callback0) /* Callback function */</pre>

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	<pre>/* For Bus No.0 */ #define COMMS_I2C_CFG_BUS0_DRIVER_TYPE #define COMMS_I2C_CFG_BUS0_DRIVER_CH</pre>	(COMMS_DRIVER_I2C) /* Driver type of I2C Bus */ (0) /* Channel No. */
I2C bus is used	<pre>/* For Bus No.1 */ #define COMMS_I2C_CFG_BUS1_DRIVER_TYPE #define COMMS_I2C_CFG_BUS1_DRIVER_CH</pre>	<pre>(COMMS_DRIVER_I2C) /* Driver type of I2C Bus */ (1) /* Channel No. */</pre>



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)

# Operation mode of ZMOD4XXX Sense	sor0	IAQ 2nd Gen.
# I2C Communication device No. for Z	MOD4XXX sensor device0	I2C Communication Device1
# I2C callback function for ZMOD4XXX	(sensor device0	zmod4xxx_user_i2c_callback0

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

	✓ Image: Configurations	
The same I2C	# Parameter Checking	System Default
bus is shared	# Number of I2C Shared Buses	1
	# Number of I2C Communication Devices	2
	✓ <sup>⊕</sup> Configurations	
The different I2C	<ul> <li>✓</li></ul>	System Default
The different I2C bus is used	<ul> <li>Configurations</li> <li>Parameter Checking</li> <li>Number of I2C Shared Buses</li> </ul>	System Default 2



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.

	# 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
The same I2C	# Timeout for the blocking bus of I2C Communication Device	OxFFFFFFF
bus is shared	# 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	OxFFFFFFF
	# I2C Shared Bus No. for I2C Communication Device0	I2C Shared Bus0
	# Slave address for I2C Communication Device0	0x44
	<ul> <li># Slave address for I2C Communication Device0</li> <li># Address mode for I2C Communication Device0</li> </ul>	0x44 7bit address mode
	Slave address for I2C Communication Device0     Address mode for I2C Communication Device0     Callback function for I2C Communication Device0	0x44 7bit address mode rm_hs300x_callback0
The different I2C	<ul> <li># Slave address for I2C Communication Device0</li> <li># Address mode for I2C Communication Device0</li> <li># Callback function for I2C Communication Device0</li> <li># Timeout for the blocking bus of I2C Communication Device</li> </ul>	0x44 7bit address mode rm_hs300x_callback0 0xFFFFFFFF
The different I2C bus is used	<ul> <li># Slave address for I2C Communication Device0</li> <li># Address mode for I2C Communication Device0</li> <li># Callback function for I2C Communication Device0</li> <li># Timeout for the blocking bus of I2C Communication Device</li> <li># I2C Shared Bus No. for I2C Communication Device1</li> </ul>	0x44 7bit address mode rm_hs300x_callback0 0xFFFFFFFF I2C Shared Bus1
The different I2C bus is used	<ul> <li># Slave address for I2C Communication Device0</li> <li># Address mode for I2C Communication Device0</li> <li># Callback function for I2C Communication Device0</li> <li># Timeout for the blocking bus of I2C Communication Device</li> <li># I2C Shared Bus No. for I2C Communication Device1</li> <li># Slave address for I2C Communication Device1</li> </ul>	0x44 7bit address mode rm_hs300x_callback0 0xFFFFFFF I2C Shared Bus1 0x32
The different I2C bus is used	<ul> <li># Slave address for I2C Communication Device0</li> <li># Address mode for I2C Communication Device0</li> <li># Callback function for I2C Communication Device0</li> <li># Timeout for the blocking bus of I2C Communication Device</li> <li># I2C Shared Bus No. for I2C Communication Device1</li> <li># Slave address for I2C Communication Device1</li> <li># Address mode for I2C Communication Device1</li> </ul>	0x44 7bit address mode rm_hs300x_callback0 0xFFFFFFFF I2C Shared Bus1 0x32 7bit address mode
The different I2C bus is used	<ul> <li># Slave address for I2C Communication Device0</li> <li># Address mode for I2C Communication Device0</li> <li># Callback function for I2C Communication Device0</li> <li># Timeout for the blocking bus of I2C Communication Device</li> <li># I2C Shared Bus No. for I2C Communication Device1</li> <li># Slave address for I2C Communication Device1</li> <li># Address mode for I2C Communication Device1</li> <li># Callback function for I2C Communication Device1</li> </ul>	0x44 7bit address mode rm_hs300x_callback0 0xFFFFFFF I2C Shared Bus1 0x32 7bit address mode rm_zmod4xxx_callback0

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.

	# I2C Driver Type for I2C Shared Bus0	SCI IIC
	# Channel No. for I2C Shared Bus0	2
The different	# Timeout for the bus lock of I2C Shared Bus0	0xFFFFFFFF
I2C bus is used	# 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	OxFFFFFFF

Г



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".

	nent				New Component		– 🗆 X
ftware Comp	conent Selection ent from those available in lis	t			Add new configuratio	n for selected component	
ategory All				~	IIC Communication (M	laster mode)	
Inction Com	munications			~	Resource:	IICA0	~
Components IIC Commun IIC Commun SPI (CSI) Co UART Comm JShow only la escription This is a clocke ay using two li	nication (Master mode)     nication (Slave mode)     mmunication     munication     stest version     sd communication function (     ines: serial clock (SCL) and se	Master mode) to comm	Type Code Generator Code Generator Code Generator Code Generator	Version 1.4.0 1.3.0 1.3.0 1.4.0 devices		IICO1 IIC10 IIC10 IIC20 IIC21 IIC30 IIC31 IIC30 IIC31	
ownload RL78 ownload ELCL onfigure gener	1 Software Integration System .modules ral settings	modules		~			



## 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.

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 <u>mutex</u> for bus lock if a recursive <u>mutex</u> 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 <u>mutex</u> for bus lock if a recursive <u>mutex</u> 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	<pre>const unsigned long long bsp_delay_time[] = {     1,     1000,     1000000 };</pre>
After	<pre>extern const unsigned long long bsp_delay_time[];</pre>

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.



Г

## 3.1.2 Add initialization process and main process

Overwrite the C file where main() or hal\_entry() is declared from the sample project.

✓
> 🔊 Includes
> 😕 ra
> 🤔 ra_gen
v 🖉 src Overwrite
> _c hal_entry.c
> LC RA_HS300X.c
> C RA_ZMOD4410.c
> 🥭 Debug
> 🧭 ra_ctg
> 🔁 script
x combination_project Debug_Platiaunch
Developer Assistance
M 19 DEVEloper Assistance
> ** /(1+)-
) 🔊 Includes
) 🖉 ra
> 😕 ra_gen
✓ 2 src
> 🖻 hal_entry.c
> C RA_HS300X.c
> 🔁 Debug
> 🗁 ra_cfg
> 🗁 script
鬱 configuration.xml
HS300x_RA6M4_NonOS Debug_Flat.jlink
K HS300x_RA6M4_NonOS Debug_Flat.launch
JLinkLog.log
R7FA6M4AF3CFB.pincfg
ra_ctg.txt
> V Developer Assistance
s 🔊 Includes
> 😝 ra gen
V 🖉 src
> c hal entry.c
> RA_ZMOD4410.c
> 🔁 Debug
> 🧀 ra_cfg
> 🧁 script
鬱 configuration.xml
JLinkLog.log
R7FA6M4AF3CFB.pincfg
📄 ra_cfg.txt
ZMOD4410_RA6M4_NonOS Debug_Flat, jlink
X ZMOD4410_RA6M4_NonOS Debug_Flat.launch
> ⑦ Developer Assistance



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.

finclude "hel_date.h"	
rsp_pre_MEADER void & ERS MarmStart(bsp_warm_start_event_t event); SSP_CPP_POTER	
void g_comms_i2c_bus0_quick_setup(void); void demo err(void):	#include "hal_data.h"
void g_hs300x_sensor0_quick_setup(void); void start_hs300x_demo(void);	FSF COP_HEDDEX void R_BSP WarmStart(bsp_warm_start_event_t event); FSP_COP_HOUTEX
vold g_mod4xxx, sensor@_guick_setup(vold); vold start_zmod4xx0_demo(vold); Copy & paste	<pre>void g_comms_i2c_bus0_quick_setup(void); void demo_err(void);</pre>
<pre>* main() is generated by the RA Configuration editor and is used to generate threads if an RTOS is used. This function * is called by main() when no RTOS is used.</pre>	<pre>void g_rmod4xxx_sensor0_quick_setup(void); void start_zmod4410_demo(void);</pre>
<pre>void bal_metry(void) {</pre>	/*************************************
/* Open HS300X */ g_hs300x_sensor0_quick_setup(); Copy & paste	/* TORD: add your own code here */ /* Open the Bus */ g_come.tac.buse_guick_setup();
<pre>/* Baset 2000 senser (active loo). Flass charge to the D0 pert connected to the RLS_H pin of the 2000 sensor on the customer board. */</pre>	<pre>/* Asset 2000 Annual (action has) Planar damp to the 10 point connected to the MS_N pla of the 2000 sensor on the customer haard. */ 1 2000 Type:relet(dion=(tr.(, SD-7), SD-74, API, 12, SD-70, Left(_MCG)); 1 2000 Type:relet(dion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2 1 2000 Type:relet(d_ion=(tr.(, SD-70, API, API, 12, SD-70, Left(_MCG)); 1 2 1 2000 Type:relet(d_ion=(tr.(, API, API, API, Left(_API, API, 12, SD-70, Left(_MCG)); 1 2 1 2000 Type:relet(d_ion=(tr.(, API, API, API, API, Left(_API, API, API, API, Left(_API, API, API, API, Left(_API, API, API, API, API, API, API, API,</pre>
المالة:() ( عدار المالغة: (Copy & paste للاستر المالغة: ( Copy & paste	
/* Entr non-iscure code */ 	14 f 60 <u>TT 55000 BUILD</u> /* Tinte mostare code */ R_BSP ManSecureEnter(); rendi
eouar }	3

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.

ttings	Property	Value	Settings	Property	Value
PI Info	✓ Common		API Info	✓ Common	
	Parameter Checking	Default (BSP)		Parameter Checking	Default (BSP)
	Pin Output Support	Disabled		Pin Output Support	Disabled
	Write Protect Enable	Disabled		Write Protect Enable	Disabled
	Clock Source	PCLKD		Clock Source	PCLKD
	<ul> <li>Module hs300x_delay Timer, General PWM (r_gpt)</li> </ul>			<ul> <li>Module zmod4410_delay Timer, General PWM (r_gp</li> </ul>	
	✓ General			✓ General	
	Name	hs300x_delay		Name	zmod4410_delay
	Channel	0		Channel	1
	Mode	Periodic		Mode	Periodic
	Period	100		Period	100
	Period Unit	Microseconds		Period Unit	Microseconds
	> Output			> Output	
	> Input			> Input	
	> Interrupts			> Interrupts	
	> Extra Features			> Extra Features	
	✓ Pins			✓ Pins	
	GTIOCOA	<unavailable></unavailable>		GTIOC1A	<unavailable></unavailable>
	GTIOCOB	<up>kunavailable&gt;</up>		GTIOC1B	<unavailable></unavailable>

## 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

```
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();
        }
Before
        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;
        }
        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();
        }
After
        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

```
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)
Before
        {
            bool wait;
            if (gs_zmod4410_delay_count > 0)
            {
               wait = true;
            }
            else
            {
               /* Stop timer */
               R_TAU0_Channel0_Stop();
               wait = false;
            }
            return wait;
        }
        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)
After
        {
            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

```
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)
Before
        {
            bool wait;
            if (gs_zmod4410_delay_count > 0)
            {
               wait = true;
            }
            else
            {
                /* Stop timer */
                R_Config_TAU0_0_Stop();
               wait = false;
            }
            return wait;
        }
        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)
After
        {
            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 <u>3.2.1 Add initialization process and main process</u>.

#### 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]  $\rightarrow$  [Settings] in the "Properties" window to open the "Settings" panel.





Select [Compiler]  $\rightarrow$  [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.

V So Common       Include file directories (-1)       Include	
StrClNSTALL/inc         Miscellaneous         Miscellaneous         Source         Language         Optimization         Missellaneous         Miscellaneous         Miscellaneous         Miscellaneous         Include files at head of compiling units (-preinclude)         Miscellaneous         Miscellaneous         Miscellaneous         Miscellaneous         Miscellaneous         Miscellaneous         Miscellaneous         Optimization         Source         Source         Object         Optimization         Optimization         Optimization         Miscellaneous	
Soligicat         Soligication         Soligication </td <td></td>	
<sup>(2)</sup> Optimization <sup>(2)</sup> Optimization <sup>(2)</sup> Miscellaneous	
<sup>(A)</sup> Miscellaneous <sup>(A)</sup> Include files at head of compiling units (-prenclude) <sup>(A)</sup> Include files at head of compiling units (-prenclude) <sup>(A)</sup> Miscellaneous <sup>(A)</sup> Include files at head of compiling units (-prenclude) <sup>(A)</sup> Include files at head of compiling units (-prenclude) <sup>(A)</sup> Miscellaneous <sup>(A)</sup> Include files at head of compiling units (-prenclude) <sup>(A)</sup> Include files at head of compiling units (-prenclude) <sup>(A)</sup> Miscellaneous <sup>(A)</sup> Miscellaneous <sup>(A)</sup> Include files at head of compiling units (-prenclude) <sup>(A)</sup> Miscellaneous <sup>(A)</sup> Miscellaneous <sup>(A)</sup> Miscellaneous	
Wer         ✓ Work         ✓ Source         Work         Work         Work         Work         Windtain         Windtain         Windtain	
Abdition     Addition	
🖄 Optimization	
💑 User Macro definition (-D) 🕢 🔬 🛞 진 🔄	
v ⊗ Linker v ⊗ Input	
🙆 Advanced	
Optimization     Section	
🖉 Device	
v ⋛ Output ≧ Advanced v	
Geffender selection — 🗆 🗙	
Select one or more Workspace Folders	
> 16 ZMOD4410_RL78G14_NonOS	
✓ I ZMOD4410 RL78G1G_NonOS	
> enerate	
Add directory path	
ZMOD4410_RX65N_NonOS	
Directory: ZMOD4410_ZMOD4510_RA6M4_Azure	
ZMOD4410_ZMOD4510_RX65N_Azure	
ZMOD4410_ZMOD4510_RX65N_FreeRTOS	
Add subdirectories	
OK Cancel Workspace File system (?) OK Cancel	
Settings 🗘	• = +
Configuration: HardwareDebug [Active]	irations
🛞 Tool Settings Toolchain Device 🎤 Build Steps 🤗 Build Artifact 🔛 Rinary Parcere 👩 Frrnr Parcere	
V 🐌 Common Include file directories (-1) 🔬 📦 🗑	
CPU     S(S(TCINSTALL/inc     S(S(TCINS	
Miscellaneous     Miscellaneous     Signa Kapace loc/SiProjName)/src/zmodAxxz_t/tib/S3-core/aq     Wis Compiler     Signa Kapace loc/SiProjName)/src/zmodAxxz_t/tib/S3-core/aq     Visite Compiler     Signa Kapace loc/SiProjName)/src/zmodAxxz_t/tib/S3-core/aq	_2nd_
Source     Syvorkspace [oc/\$[ProjName]/srd/zmod4xxv/r zmod4xxv/r]     Distance     Styvorkspace [oc/\$[ProjName]/srd/zmod4xxv/r zmod4xxv/r]     Source     Source     Syvorkspace [oc/\$[ProjName]/srd/zmod4xxv/r]     Source     Source     Syvorkspace [oc/\$[ProjName]/srd/zmod4xxv/r]     Source	fur_od g/ccrl}*
Object     ** Siworkspace loc/S[ProjName]/src/zmod4xxx/rzmod4xxx_r/lib/S3-core/rag     Dimitation     ** 'Siworkspace loc/S[ProjName]/src/zmod4xxx/inc]*	i/iar-ei
2월 Output Code 2월 Miscellaneous Include files at head of compiling units (-preinclude) 🔬 紀 🛬	51 윤1
診 MISRA C Rule Check 診 User	
Source	
Directory:	
S(workspace loc:/S(ProiName)/src)     Miscellaneous	
Wiser     Warro definition (-D)     N     N	한 문
V 🙆 Input 🕸 Advanced	
Add subdirectories	
Section Device	



Select [Compiler]  $\rightarrow$  [Source]  $\rightarrow$  [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.

Settings		<-> <-> <->
Configuration: HardwareDebug [ A	Active ]	V Manage Configurations
Nool Settings Toolchain Device	ce 🎤 Build Steps 🚇 Build Artifact 🐻	Binary Parsers 😣 Error Parsers
<ul> <li>Common</li> <li>CPU</li> <li>Device</li> <li>Miscellaneous</li> <li>Source</li> <li>Opject</li> <li>Optimization</li> <li>Output Code</li> <li>Miscellaneous</li> <li>MISRA C Rule Check</li> <li>User</li> </ul>	Check function without prototype decla Maximum size of a variable (-large_variable) Allow nested comments (-nest_commer Character encoding (-character_set) Language standard of C language (-lang) Compile strictly according to the standar	ration (-refs_without_declaration) 0x7fff bytes v nt) UTF-8 v C99 language standard v rds (-strict_std)



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

Select [Linker]  $\rightarrow$  [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]  $\rightarrow$  [Settings] in the "Properties" window to open the "Settings" panel.





(¬ → ¬ ? Settings ^ 🗸 🛞 Common Library configuration (-lang) C99 🖄 CPU ✓ runtime: Runtime routines (-head=runtime) 🖄 PIC/PID ctype.h (C89/C99): Character classification routines (-head=ctype) Miscellaneous nath.h (C89/C99): Numerical calculation library (-head=math) 🗸 🛞 Compiler mathf.h (C89/C99): Numerical calculation library (float type function) (-head=mathf) 🗸 🖄 Source 🖄 Advanced stdarg.h (C89/C99): Variable argument functions (-head=stdarg) 🖄 Object ✓ stdio.h (C89/C99): Input/Output (-head=stdio) 🦉 List ✓ stdlib.h (C89/C99): General purpose library features (-head=stdlib) string.h (C89/C99): String handling operations (-head=string) 🖄 Advanced ios (EC++): Input/Output streams (-head=ios) 🖄 Output MISRA C Rule Check new (EC++): Memory allocation and deallocation routines (-head=new) 🖄 Miscellaneous complex.h (EC++): Complex number operations (-head=complex) 🖄 User string (EC++): String manipulation operations (-head=cppstring) complex.h (C99): Complex number operations (-head=C99\_complex) 🖄 Source fenv.h (C99): Floating point operations (-head=fenv) 🖄 Object inttypes.h (C99): Integer type format operations (-head=inttypes) 🚳 List Optimization wchar.h (C99): Wide character operations (-head=wchar) 🖄 Miscellaneous wctype.h (C99): Wide character classification routines (-head=wctype) 🖄 User Enable all 🗸 🛞 Linker 🗸 🖄 Input 🖄 Advanced V 🖄 Output 🖄 Advanced 🖄 List 🖄 Optimization ✓ A Section 🖄 Symbol file 🖄 Advanced 🖄 Subcommand file 🖄 Miscellaneous 🖄 User ✓ 
S Library Generator 🖄 Mode 🖄 Standard Library 🖄 Object 🗸 🖄 Optimization Advanced Miscellaneous 👰 User

Select [Library Generator]  $\rightarrow$  [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 2<sup>nd</sup> Gen as an example.

item	content		
Board	RTK7EKA6M4S00001BE (EK-RA6M4)		
MCU	RA6M4 (R7FA6M4AF3CFB :144pin)		
Clock	200MHz		
Voltage	5V		
Environment	e <sup>2</sup> 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_laq2ndGenDataCalculate	3.2us (in stabilization)
	637us (after stabilization is complete)
RM_ZMOD4XXX_StatusCheck	3us
RM_ZMOD4XXX_DeviceErrorCheck	2.5us



## **Revision History**

		Description	
Rev.	Date	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 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

- 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.
- 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.
- This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
   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
- (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: <a href="http://www.renesas.com/contact/">www.renesas.com/contact/</a>.