Contents

List of Figures

List of Tables

1. Introduction

The ZMOD4410 Gas Sensor Module is highly configurable to meet various application needs. This document describes the general program flow to set up ZMOD4410 Gas Sensor Modules for gas measurements in a customer environment. The corresponding firmware package is provided on the Renesas product webpage at ZMOD4410 under the Downloads section.

This document also describes the function of example code provided as C code, which can be executed using the ZMOD4410 evaluation kit (EVK). For instructions on assembly, connection, and installation of the EVK hardware and software, see the document titled *ZMOD4410 Evaluation Kit Description* at ZMOD4410-evk.

The ZMOD4410 has several methods of operation available:

- IAQ 1st Gen: The IAQ algorithm ("iaq_1st_gen") outputs total volatile organic compounds (TVOC), estimated carbon dioxide level (eCO2), and a rating for the indoor air quality (IAQ) based on traditional gas sensor algorithms. This method can be used in Continuous Operation or Low Power Operation.
- IAQ 2nd Gen: The embedded artificial intelligence (AI) algorithm ("iaq_2nd_gen") derived from machine learning outputs total volatile organic compounds (TVOC), estimated carbon dioxide level (eCO2), and a rating for the indoor air quality (IAQ). This method of operation is for more accurate and consistent sensor readings. This is the recommended operation mode for IAQ.
- Odor: Sets a control signal based on an Air Quality and outputs the Air Quality Change.
- Sulfur Odor: This semi-selective detection method for gas species allows a discrimination between sulfur odors in the air. Odors are classified as "acceptable" and "sulfur" with an intensity level.

Recommendation: Before using this manual, read the *ZMOD4410 Datasheet* and corresponding documentation on the Renesas product webpage for the ZMOD4410 at ZMOD4410.

2. Requirements on Hardware to Operate ZMOD4410

To operate the ZMOD4410, customer-specific hardware with a microcontroller unit (MCU) is needed. Depending on the sensor configuration and on the hardware itself, the requirements differ and the following minimum requirements are provided as an orientation only:

- 12 to 20 kB program flash for ZMOD4410-related firmware code (MCU architecture and compiler dependent), note also Table 1
- 1kB RAM for ZMOD4410-related operations, note also Table 1
- Capability to perform I2C communication, timing functions, and floating-point instructions
- The algorithm functions work with variables saved in background and need memory retention between each call

Table 1. Exemplary Memory Footprint of ZMOD4410 Implementation on a RL78-G13 MCU

	IAQ 2nd Gen	IAQ 1st Gen	Odor	Sulfur Odor
Program flash usage in kB	13.2	10.9	8.7	9.7
RAM usage in Bytes	328	284	168	256

The ZMOD4410 firmware can be downloaded from the product webpage. To get access to the firmware a Software License Agreement has to be accepted. The firmware uses floating-point calculations with various integer and floating-point variables. A part of the firmware are precompiled libraries for many standard targets (microcontrollers), as listed in Table 2.

Table 2. Targets and Compilers Supported by Default

Target	Compiler
Arm Cortex-A	arm-none-eabi-gcc (all others)
	iar-ew-arm (IAR Embedded Workbench)
Arm Cortex-M	armcc (Keil MDK)
	armclang (Arm Developer Studio)
	arm-none-eabi-gcc (all others)
	iar-ew-arm (IAR Embedded Workbench)
	iar-ew-synergy-arm (IAR Embedded Workbench)
Arm Cortex-R4	arm-none-eabi-gcc (all others)
	iar-ew-arm (IAR Embedded Workbench)
Espressif ESP	xtensa-esp32-elf-gcc
	xtensa-lx106-elf-gcc
Microchip ATmega32	avr-gcc (AVR-Studio, AVR-Eclipse)
Microchip PIC	xc8-cc (MPLAB)
Raspberry PI	arm-linux-gnueabihf-gcc
Renesas RL78	ccrl (e^{2}studio, CS+)
	iar-ew-rl (IAR Embedded Workbench)
	rl78-elf-gcc
Renesas RX	ccrx (e^{2}studio, CS+)
	iar-ew-rx (IAR Embedded Workbench)
	rx-elf-gcc
Texas Instruments MSP430	msp430-elf-gcc
Windows	mingw32
Intel 8051	iar-ew-8051 (IAR Embedded Workbench)

Note: For other platforms (e.g., Arduino, Linux), please contact Renesas Technical Support.

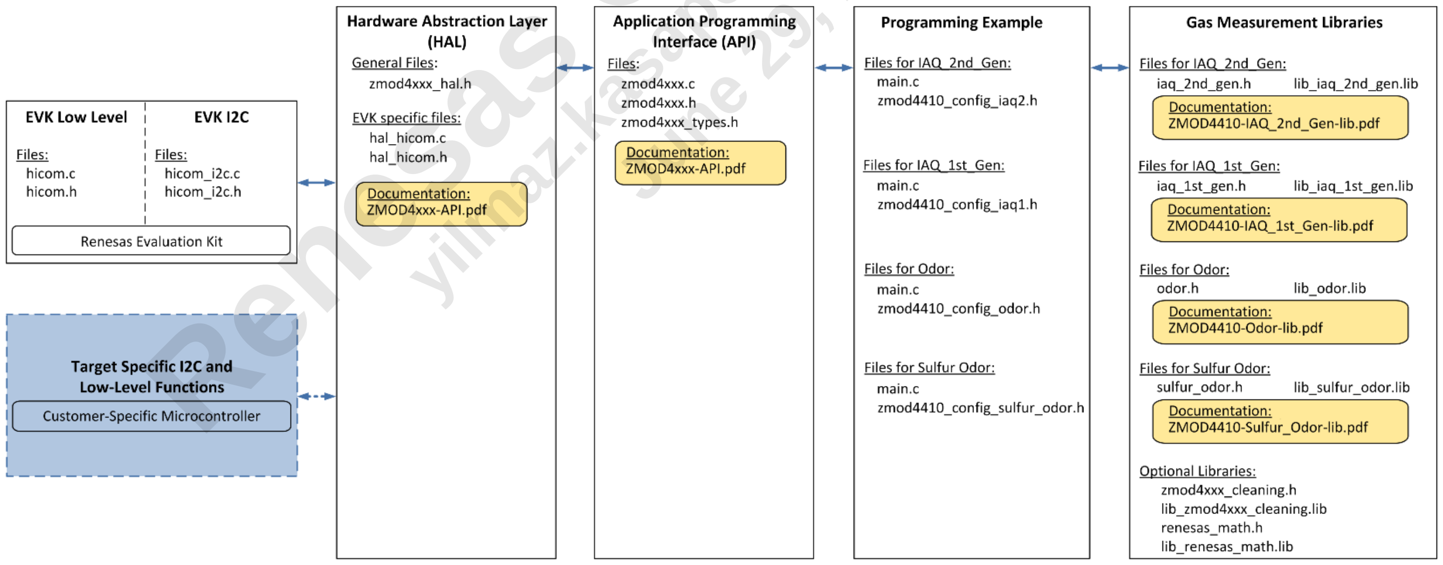
3. Structure of ZMOD4410 Firmware

To operate the ZMOD4410 and use its full functionality, five code blocks are needed as illustrated in Figure 1:

- The “Target Specific I2C and Low-Level Functions” block is the hardware-specific implementation of the I2C interface. This block contains read and write functions to communicate with the ZMOD4410 and a delay function. If the Renesas EVK is used, files for the EVK HiCom Communication Board are provided with the ZMOD4410 firmware packages. Using the user’s own target hardware requires implementing the user’s target-specific I2C and low-level functions (highlighted in light blue in Figure 1).
- The “Hardware Abstraction Layer (HAL)” block contains hardware-specific initialization and de-initialization functions. If the Renesas EVK is used, files for the EVK HiCom Communication Board are provided with the ZMOD4410 firmware packages. They need to be adjusted to the target hardware of the user. The HAL is described in the document *ZMOD4xxx-API.pdf*, which is included in the firmware packages.
- The “Application Programming Interface (API)” block contains the functions needed to operate the ZMOD4410. The API should not be modified! A detailed description of the API is located in the document *ZMOD4xxx-API.pdf*, which is included in the firmware packages.
- The “Programming Example” block provides a code example as *main.*c file that is used to initialize the ZMOD4410, perform measurements, display the data output for each specific example, and start the optional cleaning function. Each example contains one configuration file (*zmod4410_config_xxx.h*) that should not be modified! More information is provided in section 4 of this document.
- The “Gas Measurement Libraries” block contains the functions and data structures needed to calculate the firmware specific results for the Indoor Air Quality related parameters, such as IAQ, TVOC, eCO2 (IAQ 1st Gen and IAQ 2nd Gen) or Air Quality Change (Odor), or Sulfur Odor result. These algorithms cannot be used in parallel. This block also contains the optional cleaning procedure and the optional math library for code size reduction. The libraries are described in more detail in the documents *ZMOD4410-IAQ_xxx_Gen-lib.pdf*, *ZMOD4410-Odor-lib.pd*f, and *ZMOD4410-Sulfur_Odor-lib.pdf*. All of these files are part of the downloadable firmware packages.

To avoid naming conflicts, all API function names start with the prefix “`zmod4xxx_`” in the ZMOD4410 code. This naming applies to all operation methods of the ZMOD4410.

Figure 1. File Overview for ZMOD4410 Firmware



All files are part of zipped firmware packages available on www.IDT.com/zmod4410 in the Downloads section. Please be aware that not all configurations and optional libraries are available for all operation methods but the individual library documentation will provide detailed insight on possible settings.

4. Description of the Programming Examples

This section describes the programming examples and how to use the ZMOD4410 Gas Sensor Module. In the examples, the ZMOD4410 is initialized, the measurement is started, and measured values are outputted. The examples are intended to work on a Windows® computer in combination with the Renesas Gas Sensor EVK but can be easily adjusted to operate on other platforms.

To run each example using the EVK without further configuration, start the files *zmod4410_xxx_example.exe,* which are included in the firmware packages.

4.1 IAQ 1st Gen Example

The *main.c* file of the example contains the main program flow. To switch between the modes of operation (Continuous and Low Power), define the compiler switch ZMOD_CONF_TYPE in *main.c* to CONTINUOUS_MODE or LOW_POWER_MODE.

First, the target-specific initializations are performed in the example. The ZMOD4410 is configured by reading device parameters and Final Module Test parameters from the sensor's nonvolatile memory (NVM) and initializing it to run at its operating temperature. An endless loop continuously checks the status of the ZMOD4410 and reads its data. The raw data is subsequently processed, and the TVOC, IAQ, eCO2 algorithm results are calculated. All values are printed in the command line window. To stop the loop, press any key, which releases the hardware and stops the program. For more details, refer to the example code.

Table 3. IAQ 1st Gen Program Flow for Continuous and Low Power Mode

Note: Blue shaded lines can be run in an endless loop with polling or interrupt usage. Steps in brackets are only needed for Low Power Mode.

Line	Program Actions	Notes	API and Algorithm Functions
1	Reset the sensor.	Before configuring the sensor, reset the sensor by powering it off/on or toggling the reset pin.	-
2	Read device parameters from the nonvolatile memory (NVM).	This step is required to select the correct configuration for the sensor.	zmod4xxx_read_sensor_info
3	Prepare the sensor for IAQ measurements.	This function must be called after every startup.	zmod4xxx_prepare_sensor
4	Initialize the IAQ (TVOC, eCO2) algorithms.	Read calibration data and initialize algorithms.	init_iaq_1st_gen
5	Start the measurement.	Measurement runs in an endless loop for Standard Mode. Single Measurement is started in Low Power Mode.	zmod4xxx_start_measurement
6	Read status register.	Wait until the measurement is done. This will also be signaled on the interrupt pin with a falling signal (edge detection needed).	zmod4xxx_read_status
7	Read sensor ADC output.	Result contains raw sensor output.	zmod4xxx_read_adc_results
8	Algorithm calculation.	Calculate current MOx resistance Rmox, IAQ, TVOC, and eCO2. First 10 samples are ignored for sensor stabilization.	calc_iaq_1st_gen
(9)	Delay (5475ms).		
(10)	Start next measurement.	One measurement is started for Low Power Mode.	zmod4xxx_start_measurement

4.2 IAQ 2nd Gen Example

The *main.c* file of the example contains the main program flow.

First, the target-specific initializations are performed. The ZMOD4410 is configured by reading device parameters as well as Final Module Test parameters from the sensor's nonvolatile memory (NVM) and initializing it to run a sequence of different operating temperatures. An endless loop continuously checks the status of the ZMOD4410 and reads its data. The raw data is subsequently processed, and the TVOC, IAQ, eCO2 algorithm results are calculated with the embedded neural net. All values are printed in the command line window. To stop the loop, press any key, which releases the hardware and stops the program. For more details, refer to the example code.

Table 4. IAQ 2nd Gen Program Flow

Note: Blue shaded lines can be run in an endless loop with polling or interrupt usage.

Line	Program Actions	Notes	API and Algorithm Functions
1	Reset the sensor.	Before configuring the sensor, reset the sensor by powering it off/on or toggling the reset pin.	-
2	Read device parameters from the nonvolatile memory (NVM).	This step is required to select the correct configuration for the sensor.	zmod4xxx_read_sensor_info
3	Prepare the sensor for IAQ measurements.	This function must be called after every startup.	zmod4xxx_prepare_sensor
4	Initialize the IAQ (TVOC, eCO2) algorithms.	Read calibration data and initialize algorithms.	init_iaq_2nd_gen
5	Start the measurement.	One measurement is started.	zmod4xxx_start_measurement
6	Read status register.	Wait until the measurement is done. This will also be signaled on the interrupt pin with a falling signal (edge detection needed).	zmod4xxx_read_status
7	Read sensor ADC output.	Result contains raw sensor output.	zmod4xxx_read_adc_results
8	Algorithm calculation.	Calculate current MOx resistance Rmox, IAQ, TVOC, and eCO2. First 10 samples are ignored for sensor stabilization.	calc_iaq_2nd_gen
9	Delay (1990ms).		
10	Start next measurement.	One measurement is started.	zmod4xxx_start_measurement

4.3 Odor Example

The *main.c* file of the example contains the main program flow. Odor control state and Air Quality Change.

First, the target-specific initializations are performed in the example. The ZMOD4410 is configured by reading device parameters as well as Final Module Test parameters from the sensor's nonvolatile memory (NVM) and initializing it to run at its operating temperature. An endless loop continuously checks the status of the ZMOD4410 and reads its data. The raw data is subsequently processed, and the Odor control state and Air Quality Change algorithm results are calculated. All values are printed in the command line window. To stop the loop, press any key, which releases the hardware and stops the program. For more details, refer to the example code.

Table 5. Odor Program Flow

Note: Blue shaded lines can be run in an endless loop with polling or interrupt usage.

Line	Program Actions	Notes	API and Algorithm Functions
1	Reset the sensor.	Before configuring the sensor, reset the sensor by powering it off/on or toggling the reset pin.	-
2	Read device parameters from the nonvolatile memory (NVM).	This step is required to select the correct configuration for the sensor.	zmod4xxx_read_sensor_info
3	Prepare the sensor for odor measurements.	This function must be called after every startup.	zmod4xxx_prepare_sensor
4	Start the measurement.	Measurement runs in an endless loop.	zmod4xxx_start_measurement
5	Read status register.	Wait until the measurement is done. This will also be signaled on the interrupt pin with a falling signal (edge detection needed).	zmod4xxx_read_status
6	Read sensor ADC output.	Result contains raw sensor output.	zmod4xxx_read_adc_results
7	Get the MOx resistance value.	Get the MOx resistance value.	zmod4xxx_calc_rmox
8	Algorithm calculation.	Calculate Odor control state cs_state and Air Quality Change conc_ratio. First 10 samples are ignored for sensor stabilization.	calc_odor

4.4 Sulfur Odor Example

The *main.c* file of the example contains the main program flow.

First, the target-specific initializations are performed. The ZMOD4410 is configured by reading device parameters as well as Final Module Test parameters from the sensor's nonvolatile memory (NVM) and initializing it to run a sequence of different operating temperatures. An endless loop continuously checks the status of the ZMOD4410 and reads its data. The raw data is subsequently processed, and the Sulfur Odor algorithm results are calculated with the embedded neural net. All values are printed in a command line window. To stop the loop, press any key, which releases the hardware and stops the program. For more details, refer to the example code.

Table 6. Sulfur Odor Program Flow

Note: Blue shaded lines can be run in an endless loop with polling or interrupt usage.

Line	Program Actions	Notes	API and Algorithm Functions
1	Reset the sensor.	Before configuring the sensor, reset the sensor by powering it off/on or toggling the reset pin.	-
2	Read device parameters from the nonvolatile memory (NVM).	This step is required to select the correct configuration for the sensor.	zmod4xxx_read_sensor_info
3	Prepare the sensor for IAQ measurements.	This function must be called after every startup.	zmod4xxx_prepare_sensor
4	Initialize the Sulfur Odor algorithm.	Read calibration data and initialize algorithm.	init_sulfur_odor
5	Start the measurement.	One measurement is started.	zmod4xxx_start_measurement
6	Read status register.	Wait until the measurement is done. This will also be signaled on the interrupt pin with a falling signal (edge detection needed).	zmod4xxx_read_status
7	Read sensor ADC output.	Result contains raw sensor output.	zmod4xxx_read_adc_results
8	Algorithm calculation.	Calculate current MOx resistance Rmox, Sulfur Odor intensity, and classification. First 10 samples are ignored for sensor stabilization.	calc_sulfur_odor
9	Delay (1990ms).		
10	Start next measurement.	One measurement is started.	zmod4xxx_start_measurement

4.5 Optional Libraries

4.5.1 Cleaning Procedure

The cleaning procedure is only recommended if the user believes there is a problem with his product assembly (e.g., contamination from solder vapors). The cleaning process takes about 10 minutes and helps to clean the metal oxide surface from assembly residues. Please use *zmod4xxx_cleaning* library for this purpose. The example code shows how to use the cleaning function. The MOx resistance will usually be lower after the cleaning procedure and slowly rises over time again. Although the sensor will immediately respond to any gas concentration through a sophisticated baseline correction.

Important note: If needed, the cleaning procedure should be executed after PCB assembly during final production test and can run only once during the lifetime of each module. The cleaning function is commented out in the example to not use it by default.

4.5.2 Code Size Reduction

To decrease overall code size by approximately 1.5kB (depending on target platform), the *renesas_math* library is provided. It can replace the default math library of the target platform. The *renesas_math* library replaces the functions *pow()*, *log()* and *exp()* but makes other standard math functions unavailable. Please read the documentation for IAQ 1st Gen or Odor Method for usage of this library. It is not compatible with the IAQ 2nd Gen nor with the Sulfur Odor Method.

5. Adapting the Programming Example for Target Hardware

5.1 System Hierarchy and Implementation Steps

Renesas' ZMOD4410 C API is located between the application and the hardware level.

Figure 2. System Hierarchy

Customer Application	
Application-Specific Configuration of the Programming Example	
ZMOD4410 API and Libraries (Algorithms)	
Hardware Abstraction Layer (HAL)	
Low-Level I2C Communication	Low-Level Hardware Functions
Hardware Level (ZMOD4410 and Target)	

The low-level I2C functions are implemented in the file *hicom_i2c.c* and are allocated in the *hal_hicom.c* (see Figure 1) for the EVK hardware running on a Windows-based computer and the HiCom Communication Board. To incorporate this programming example into a different hardware platform, the following steps are necessary:

1. Establish I2C communication and conduct register test. Please find detailed hints in the "I2C Interface and Data Transmission Protocol" section of the *ZMOD4410 Datasheet*.
2. Adjust the HAL files and hardware-specific *init_hardware* and *deinit_hardware* functions to the user's target hardware (compare with *hal_hicom.c* file). Set the device's struct pointers *read*, *write,* and *delay_ms* in the hardware initialization by using wrapper functions. The type definitions of the function pointers can be found in *zmod4xxx_types.h* (see Figure 1) and an implementation example for the EVK in the *hicom_i2c.c*. The functions *read* and *write* should point to the I2C implementation of the hardware used. Test the *delay_ms* function with a scope plot.
3. Use the example code without the algorithm library functions first. Therefore, comment out all library related code (functions start with *init_* and *calc_*). Test if the adapted example runs and *zmod4xxx_read_adc_results()* function outputs changing ADC values in main measurement loop.
4. To apply the algorithms and get their output, include the corresponding library in the extra *gas-algorithm-libraries* folder. Use precompiled libraries according to target hardware-platform and IDE/compiler (see Table 2). Uncomment the corresponding functions (functions start with *init_* and *calc_*).

5.2 Error Codes

All API functions return a code to indicate the success of the operation. If no error occurred, the return code is zero. In the event of an error, a negative number is returned. The API has predefined symbols *zmod4xxx_err* for the error codes defined in *zmod4xxx_types.h*.

5.3 Interrupt Usage

The Programming Examples are written in polling mode. In this mode ZMOD4410’s interrupt pin (INT) is not used. However, depending on target hardware and application it might be beneficial to make use of the interrupt. The following interrupt usages are possible:

- Interrupt pin (INT): This pin indicates the end of a measurement sequence with a falling edge and stays LOW until the next measurement regardless if the results are read or not. Care needs to be taken for IAQ 1st Gen Continuous Mode and for Odor measurements. In these examples, the measurement timing is driven by ZMOD4410’s internal ASIC. The LOW phase after each measurement is therefore very short and an edge detection is needed. The detection of an interrupt can replace *zmod4xxx_read_status()* call and its corresponding eoc polling loop in *main.c* file.
- Timer-based interrupts: Some target hardware has the possibility to use timer-based interrupts. For this usage, the main measurement loop has to be called periodically with the corresponding measurement intervals. This procedure can replace delays, which are used to ensure measurement timing. The measurement intervals for each example are as follows.

  2 seconds: IAQ 1st Gen Continuous Mode, Odor (no delays used),

  3 seconds: IAQ 2nd Gen, Sulfur Odor (replaces the delay of 1990ms),

  6 seconds: IAQ 1st Gen Low Power Mode (replaces the delay of 5475ms).

5.4 Adaptions to Follow C90 Standard

ZMOD4410 firmware supports C99 standard and later. A few configurations changes are required to comply with versions earlier than C99.

- Initialization of a structure: C90 standard allows the members only to appear in a fixed order, the same as the array or structure was initialized. In C99 standard, you can initialize and call the elements in any order by using designators. The file *zmod4410_config_xxx.h* needs to be edited. Change all designated initializations by erasing "`.member_name =`" in structure initializations, for example:

```
typedef struct {
    char *a[3];
    char *b[3];
    char *c[3];
} test_struct;

/* C99 STANDARD */
test_struct struct_C99 = {
    .a = {"a", "b", "c"},
    .b = {"d", "e", "f"},
    .c = {"g", "h", "i"}
};

/* C90 STANDARD */
test_struct struct_C90 = {
        { "a", "b", "c" },  /* .a */
        { "d", "e", "f" },  /* .b */
        { "g", "h", "i" }   /* .c */
};
```

- *stdint.h* file: *stdint.h* is used in API and examples. However, *stdint.h* file is introduced with C99 standards. Therefore, it should be added manually when working with a standard earlier than C99. This is the content needed for *stdint.h*:

```
#ifndef STDINT_H
#define STDINT_H

typedef unsigned char uint8_t;
typedef unsigned short uint16_t;
typedef unsigned long uint32_t;
typedef uint32_t uint64_t[2];

typedef signed char int8_t;
typedef short int16_t;
typedef long int32_t;
typedef int32_t int64_t[2];

#endif
```

5.5 How to Compile for EVK Hardware

The Programming Example is written to work with the EVK hardware. To evaluate the impact of code changes on sensor performance, it is possible to use the EVK as reference. This section provides a manual to compile the adapted source code into an executable file. This executable can be used with the EVK on a Windows platform. For compiling, MinGW needs to be installed. The folder structure is identical to that in the download package. The procedure is described on the IAQ 2nd Gen Example (*iaq_2nd_gen*). To adapt it for the other examples just replace the corresponding name (*iaq_1st_gen*, *odor*, *sulfur_odor*).

1. Install MinGW:
   - MinGW (32 bit) must be used. Mingw64 will not work due to the 32-bit FTDI library for the EVK HiCom board.
   - Download *mingw-get-setup.exe* from https://osdn.net/projects/mingw/releases/.
   - The downloaded executable file installs “Install MinGW Installation Manager”.
   - Select required packages:

     mingw-developer-toolkit-bin

     mingw32-base-bin

     mingw32-gcc-g++-bin

     msys-base-bin.
   - Click “Installation” from the left-top corner and select “Update Catalogue”.
   - Finish installation.
2. Add the *mingw-gcc* in system path:
   - Open “Control Panel”, select “System”, select “Advanced System Settings”, select “Environment Variables”
   - Find “Path” in System Variables then add *C:\MinGW\bin* (change the path in case MinGW is installed in different location.)
3. Compiling:
   - Go to Command Prompt and change to the following directory of the example folder:
     [...]\Renesas_ZMOD4410_iaq_2nd_gen_Example\ZMOD4410_Firmware\zmod4xxx_example
   - Execute the following command in one line:
     ```
     gcc src\*.c HAL\*.c -o zmod4410_iaq_2nd_gen_example_custom.exe -Isrc -IHAL
     -I..\gas-algorithm-libraries\iaq_2nd_gen\Windows\x86\mingw32 -L. -l:HAL\RSRFTCI2C.lib
     -l:..\gas-algorithm-libraries\iaq_2nd_gen\Windows\x86\mingw32\lib_iaq_2nd_gen.lib
     ```
     Note, gcc command may need admin rights!
   - An executable file called *zmod4410_iaq_2nd_gen_example_custom.exe* will be created.

6. Revision History

Revision Date	Description of Change
September 24, 2020	▪ Add sections "Interrupt Usage", "Adaptions to Follow C90 Standard", "How to Compile for EVK Hardware". ▪ Add example for memory footprint and update target and compiler list. ▪ Refined implementation steps. ▪ Minor edits in text.
May 27, 2020	▪ Completed many changes throughout the document
November 14, 2019	▪ Cleaning procedure added and explained. ▪ Figure for file overview updated.
February 12, 2019	▪ Update for change in the program flow for Continuous (skip the first 10 samples) and Low Power (skip the first 5 samples) Operation Modes. ▪ Implementation of plain trim value calibration. ▪ Minor edits in text.
December 5, 2018	▪ Update for Low Power Operation. ▪ Minor edits.
September 27, 2018	▪ Revision of document title from ZMOD44xx Programming Manual with ZMOD4410 Example to ZMOD4410 Programming Manual – Read Me. ▪ Full update for Odor Operation Mode 2. ▪ Minor edits.
June 11, 2018	Initial release.

(Rev.1.0 Mar 2020)

Corporate Headquarters

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

Contact Information

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

Trademarks