ZMOD4410 - Indoor Air Quality Sensor Platform

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 ZMOD4410 product page 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) and with Arduino hardware. For instructions on assembly, connection, and installation of the EVK hardware and software, see the *ZMOD4410 Evaluation Kit User Manual* on the ZMOD4410 EVK product page.

The ZMOD4410 has several modes of operation:

- IAQ 2nd Gen – The embedded artificial intelligence (AI) algorithm (“iaq_2nd_gen”) derived from machine learning outputs total volatile organic compounds (TVOC), equivalent ethanol (EtOH) concentration, estimated carbon dioxide level (eCO2), and a rating for the indoor air quality (IAQ). This method of operation is for highly accurate and consistent sensor readings. **This is the recommended operation mode for IAQ**.
- IAQ 2nd Gen Ultra Low Power – The embedded artificial intelligence (AI) algorithm (“iaq_2nd_gen_ulp”) derived from machine learning outputs total volatile organic compounds (TVOC), equivalent ethanol (EtOH) concentration, estimated carbon dioxide level (eCO2), and a rating for the indoor air quality (IAQ). This method of operation offers a much lower power consumption while keeping accurate and consistent sensor readings.
- 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.
- IAQ 1st Gen in Continuous Operation or Low Power Operation (Legacy)

Recommendation: Before using this document, read the *ZMOD4410 Datasheet* and corresponding documentation on the ZMOD4410 product page.

Contents

Figures

Tables

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 30 kB program flash for ZMOD4410-related firmware code (MCU architecture and compiler dependent), see Table 1
- 1kB RAM for ZMOD4410-related operations (see Table 1)
- Capability to perform I^{2}C 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 Renesas RL78-G13 MCU

	IAQ 2nd Gen	IAQ 2nd Gen ULP	Odor	Sulfur Odor
Program flash usage in kB	15.8	13.8	8.7	9.7
RAM usage (required variables) in bytes	332	308	168	256
RAM usage (stack size for library functions, worst case) in bytes	500	300	64	268

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

Table 2. Targets and Compilers Supported by Default

Target	Compiler
Arduino (Cortex-M0+, ATmega32)	arm-none-eabi-gcc (Arduino IDE)
	avr-gcc (Arduino IDE)
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-esp32s2-elf-gcc
	xtensa-lx106-elf-gcc
Intel 8051	iar-ew-8051 (IAR Embedded Workbench)
Microchip ATmega32 and AVR	avr-gcc (AVR-Studio, AVR-Eclipse, MPLAB, Atmel Studio)
Microchip PIC	xc8-cc (MPLAB)
Raspberry Pi	arm-linux-gnueabihf-gcc
Renesas RL78	ccrl (e²studio, CS+)
	iar-ew-rl (IAR Embedded Workbench)
	rl78-elf-gcc
Renesas RX	ccrx (e²studio, CS+)
	iar-ew-rx (IAR Embedded Workbench)
	rx-elf-gcc
Texas Instruments MSP430	msp430-elf-gcc
Windows	mingw32

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

3. Structure of ZMOD4410 Firmware

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

- The "Target Specific I2C and Low-Level Functions" block is the hardware-specific implementation of the I^{2}C 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 I^{2}C and low-level functions (this is 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 Description of the Programming Examples.
- 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, EtOH, eCO2 (IAQ 2nd Gen and IAQ 2nd Gen ULP) or Air Quality Change (Odor), or Sulfur Odor result. These algorithms cannot be used in parallel. This block also contains the optional cleaning. The libraries are described in more detail in the documents *ZMOD4410-IAQ_2nd_Gen-lib.pdf*, *ZMOD4410-IAQ_2nd_Gen_ULP-lib.pdf*, *ZMOD4410-Odor-lib.pdf*, 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 ZMOD4410 operation modes. The Arduino examples have a similar structure, but have some other features that facilitate operation with the Arduino board (see Arduino Examples).

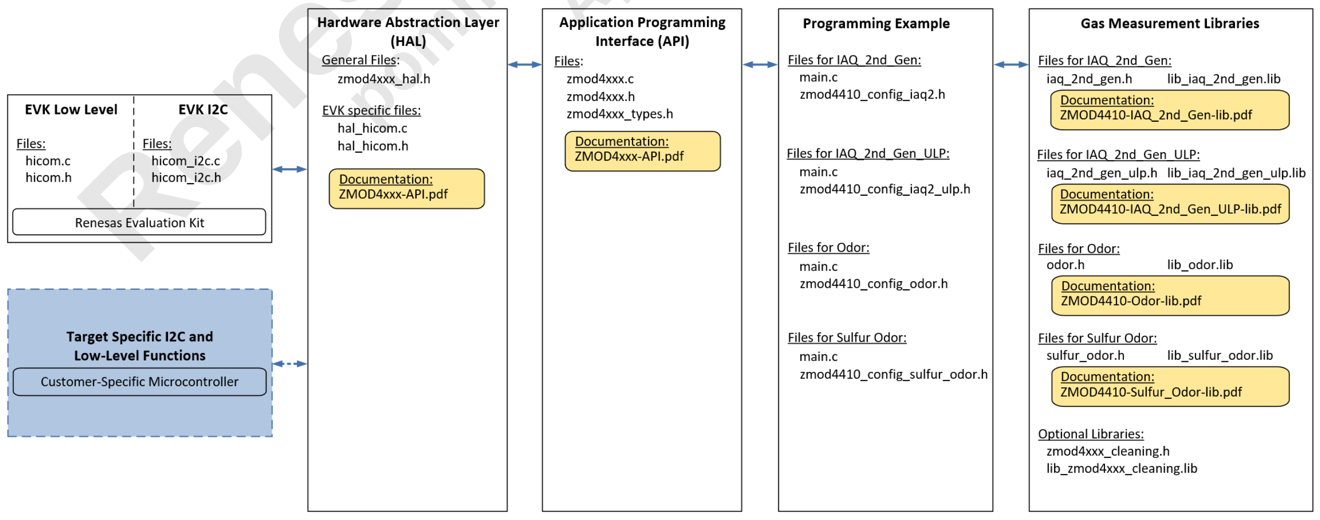


Figure 1. File Overview for ZMOD4410 Firmware

All files are part of zipped firmware packages available on the ZMOD4410 product page under the Downloads section. Note that not all configurations and optional libraries are available for all operation methods; the individual library documentation will provide detailed insight on possible settings.

4. Description of the Programming Examples

This section describes the structure of the programming examples and the steps needed to operate the sensor module. In the examples, the ZMOD4410 is initialized, the measurement is started, and measured values are outputted. They 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 (see "Adapting the Programming Example for Target Hardware"). To run each example using the EVK without further configuration, start the files *zmod4410_xxx_example.exe,* which are included in the firmware packages. Arduino examples will work for the described Arduino hardware (see "Arduino Examples").

4.1 IAQ 2nd Gen Example for EVK

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 non-volatile memory (NVM) and initializing it to run a sequence of different operating temperatures. An endless measurement loop continuously checks the status of the ZMOD4410 and reads its data. The raw data is subsequently processed, and the TVOC, EtOH, IAQ, eCO2 algorithm results are calculated with the embedded neural net machine learning algorithm. 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 information, see the example code.

Note: The blue colored lines in the following table can be run in an endless loop with polling or interrupt usage.

Table 3. IAQ 2nd Gen Program Flow (Cont. on Next Page)

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 product ID and configuration parameters.	This step is required to select the correct configuration for the sensor.	zmod4xxx_read_sensor_info
3	Calibration parameters are determined and measurement is configured.	This function must be called after every startup.	zmod4xxx_prepare_sensor
4	Initialize the IAQ (TVOC, EtOH, eCO2) algorithm.	Gas Algorithm Library function. Initialize again after 48 hours of sensor operation.	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	Check if an error occurred.	Check for an access conflict during register read-out and for a Power-On Reset.	zmod4xxx_check_error_event
8	Read sensor ADC output.	Result contains raw sensor output.	zmod4xxx_read_adc_result
9	Algorithm calculation.	Calculate current MOx resistance Rmox, IAQ, TVOC, EtOH and eCO2. First 60 samples (3 minutes) are used for minimal, hard-coded sensor stabilization. Actual stabilization can take longer (up to 48 hours).	calc_iaq_2nd_gen

Line	Program Actions	Notes	API and Algorithm Functions
10	Delay (1990ms).	This delay is necessary to keep the right measurement timing and to call a measurement every 3 seconds with a maximum deviation of 5%.	-
11	Start next measurement.	One measurement is started.	zmod4xxx_start_measurement

4.2 IAQ 2nd Gen ULP Example for EVK

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 non-volatile memory (NVM) and initializing it to run a sequence of different operating temperatures. An endless measurement loop continuously checks the status of the ZMOD4410 and reads its data. The raw data is subsequently processed, and the TVOC, EtOH, IAQ, eCO2 algorithm results are calculated with the embedded neural net machine learning algorithm. 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 information, see the example code.

Note: The blue colored lines in the following table can be run in an endless loop with polling or interrupt usage.

Table 4. IAQ 2nd Gen ULP Program Flow (Cont. on Next Page)

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 product ID and configuration parameters.	This step is required to select the correct configuration for the sensor.	zmod4xxx_read_sensor_info
3	Calibration parameters are determined and measurement is configured.	This function must be called after every startup.	zmod4xxx_prepare_sensor
4	Initialize the IAQ (TVOC, EtOH, eCO2) algorithm.	Gas Algorithm Library function. Initialize again after 48 hours of sensor operation.	init_iaq_2nd_gen_ulp
5	Start the measurement.	One measurement is started.	zmod4xxx_start_measurement
6	Delay ().	Wait until the measurement is done. This is the first delay. It should be longer than 1010 ms.	-
7	Read status register.	Check if the measurement is done. This will also be signaled on the interrupt pin with a falling signal (edge detection needed).	zmod4xxx_read_status
8	Check if an error occurred.	Check for a Power-On Reset.	zmod4xxx_check_error_event
9	Read sensor ADC output.	Result contains raw sensor output.	zmod4xxx_read_adc_result
10	Check if an error occurred.	Check for errors during ADC readout.	zmod4xxx_check_error_event

Line	Program Actions	Notes	API and Algorithm Functions
11	Algorithm calculation.	Calculate current MOx resistance Rmox, IAQ, TVOC, EtOH and eCO2. Relative humidity (in % RH) and temperature values (in °C) need to be passed as arguments. First 10 samples (15 minutes) are used for minimal, hard-coded sensor stabilization. Actual stabilization can take longer (up to 48 hours).	calc_iaq_2nd_gen
12	Delay ().	This second delay is necessary to keep the right measurement timing. The sum of the first and second delay should amount 90 seconds to call a measurement every 90 seconds with a maximum deviation of 5%.	-

4.3 Odor Example for EVK

The *main.c* file of the example contains the main program flow. 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 non-volatile memory (NVM) and initializing it to run at its operating temperature. An endless measurement 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 information, see the example code.

Note: The blue colored lines in the following table can be run in an endless loop with polling or interrupt usage.

Table 5. Odor Program Flow

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 product ID and configuration parameters.	This step is required to select the correct configuration for the sensor.	zmod4xxx_read_sensor_info
3	Calibration parameters are determined and measurement is configured.	This function must be called after every startup.	zmod4xxx_prepare_sensor
5	Start the measurement.	Measurement runs in an endless loop.	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_result
8	Get the MOx resistance value.	Get the MOx resistance value.	zmod4xxx_calc_rmox
9	Algorithm calculation.	Calculate Odor control state *cs_state* and Air Quality Change *conc_ratio*. First 15 samples (30 seconds) are used for minimal, hard-coded sensor stabilization. Actual stabilization can take longer (up to 48 hours).	calc_odor

4.4 Sulfur Odor Example for EVK

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 non-volatile memory (NVM) and initializing it to run a sequence of different operating temperatures. An endless measurement loop continuously checks the status of the ZMOD4410 and reads its data. The raw data is subsequently processed, and the Sulfur Odor intensity and classification results are calculated with the embedded neural net machine-learning algorithm. 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 information, see the example code.

Note: The blue colored lines in the following table can be run in an endless loop with polling or interrupt usage.

Table 6. Sulfur Odor Program Flow

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 product ID and configuration parameters.	This step is required to select the correct configuration for the sensor.	zmod4xxx_read_sensor_info
3	Calibration parameters are determined and measurement is configured.	This function must be called after every startup.	zmod4xxx_prepare_sensor
4	Initialize the IAQ (TVOC, EtOH, eCO2) algorithm.	Gas Algorithm Library function. Initialize again after 48 hours of sensor operation.	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_result
8	Algorithm calculation.	Calculate current MOx resistance Rmox, Sulfur Odor intensity, and classification. First 60 samples (3 minutes) are used for minimal, hard-coded sensor stabilization. Actual stabilization can take longer (up to 48 hours).	calc_sulfur_odor
9	Delay (1990ms).	This delay is necessary to keep the right measurement timing and call a measurement every 3 seconds with a maximum deviation of 5%.	-
10	Start next measurement.	One measurement is started.	zmod4xxx_start_measurement

4.5 Arduino Examples

To set up a firmware for an Arduino target, Renesas provides the above-mentioned EVK examples also as Arduino examples. These examples have a similar structure as shown in Figure 1 but with a HAL dedicated for Arduino, an Arduino-compatible structure, and Arduino-specific files. One example supports SAMD 32-bit ARM Cortex-M0+ based Arduino-Hardware included in the SAMD Boards library. For example:

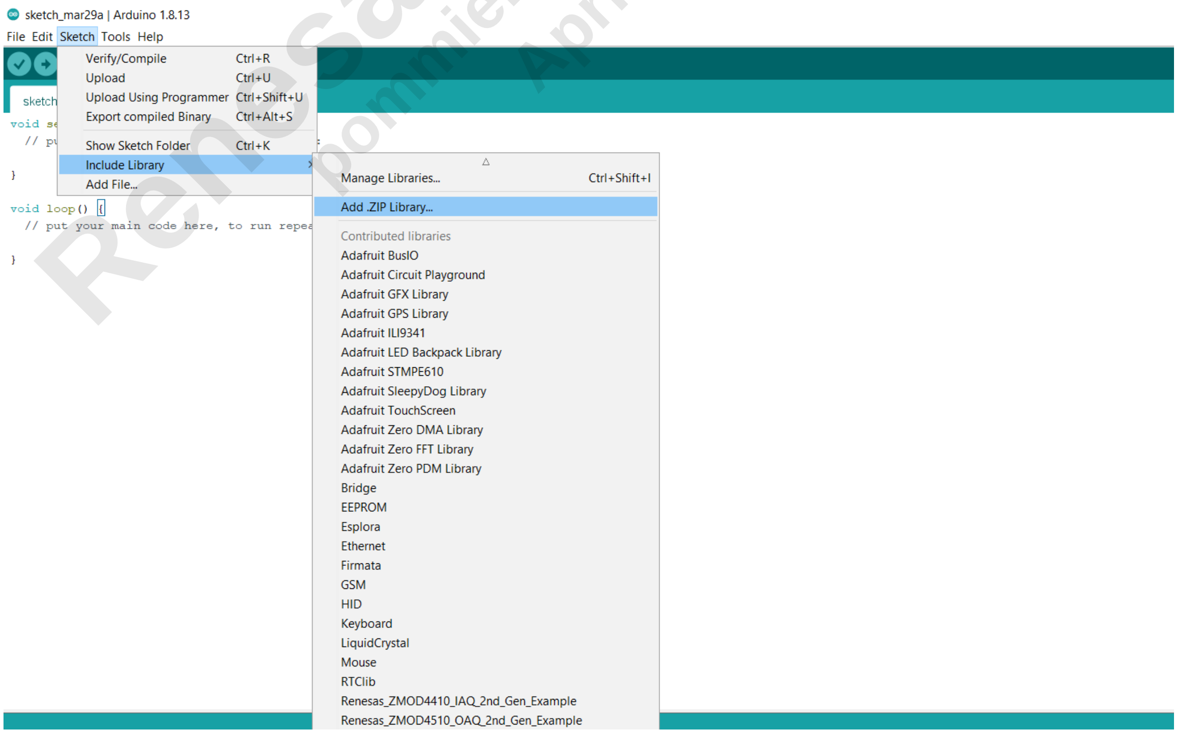
- Arduino Zero/MKR Zero
- Arduino MKR1000
- Arduino Nano 33 IoT
- Arduino M0
- etc.

The other example supports AVR ATmega32 based Arduino-Hardware included in the AVR Boards library. For example:

- Arduino Uno Rev3 (SMD)
- Arduino Nano
- Arduino Micro
- Arduino Leonardo
- etc.

The Program Flows correspond to those displayed in the EVK examples. To get the Arduino example started, complete the following steps (exemplary shown for SAMD 32-bit ARM Cortex-M0+ based boards):

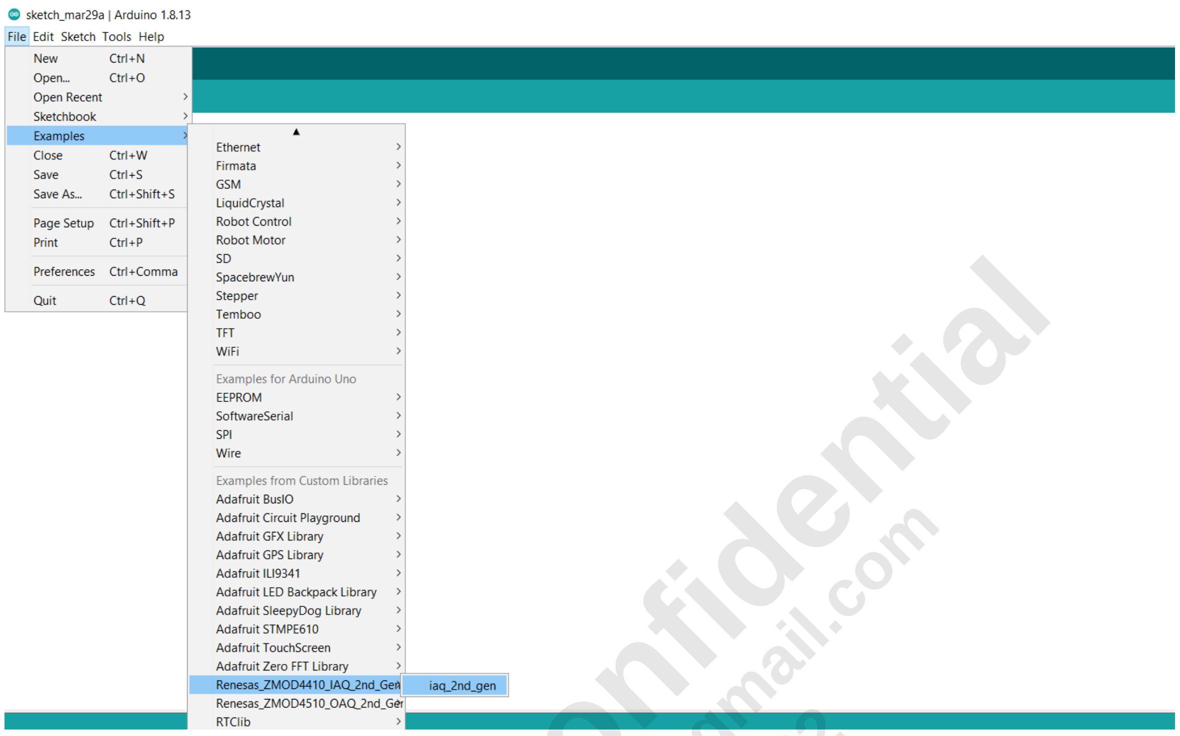
1. Connect the ZMOD4410 to the Arduino board. To connect the EVK Sensor Board, check the pin configuration on connector X1 in the *ZMOD4410 EVK User Manual* on the [ZMOD4410 EVK]product page.
2. Go to the Arduino example path (for example, […]\Documents\Arduino\libraries) and check if a ZMOD4410 example exists. Old example folders must be deleted.
3. Open Arduino IDE. Select “Sketch > Include Library > Add .ZIP library”.


sketch_mar29a | Arduino 1.8.13
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
Manage Libraries... Ctrl+Shift+I
Add .ZIP Library...
Contributed libraries
Adafruit BusIO
Adafruit Circuit Playground
Adafruit GFX Library
Adafruit GPS Library
Adafruit ILI9341
Adafruit LED Backpack Library
Adafruit STMPE610
Adafruit SleepyDog Library
Adafruit TouchScreen
Adafruit Zero DMA Library
Adafruit Zero FFT Library
Adafruit Zero PDM Library
Bridge
EEPROM
Esplora
Ethernet
Firmata
GSM
HID
Keyboard
LiquidCrystal
Mouse
RTClib
Renesas_ZMOD4410_IAQ_2nd_Gen_Example
Renesas_ZMOD4510_OAQ_2nd_Gen_Example

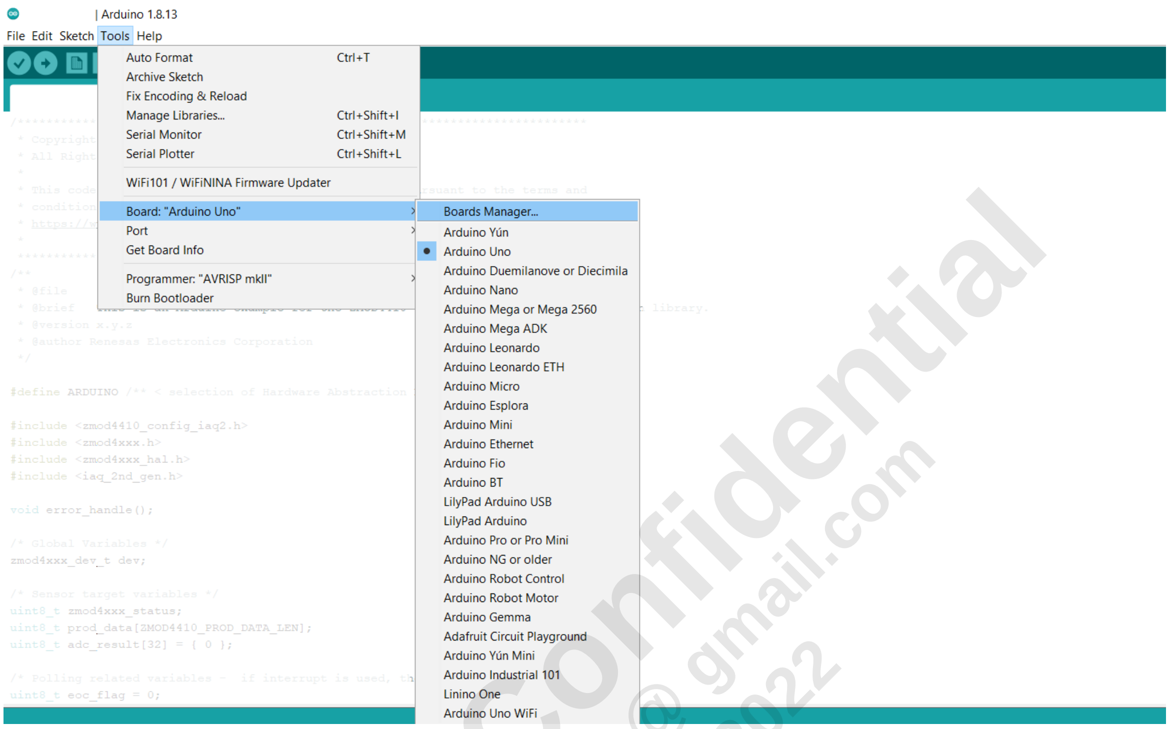
```
void loop() {
  // put your main code here, to run repea
}
```

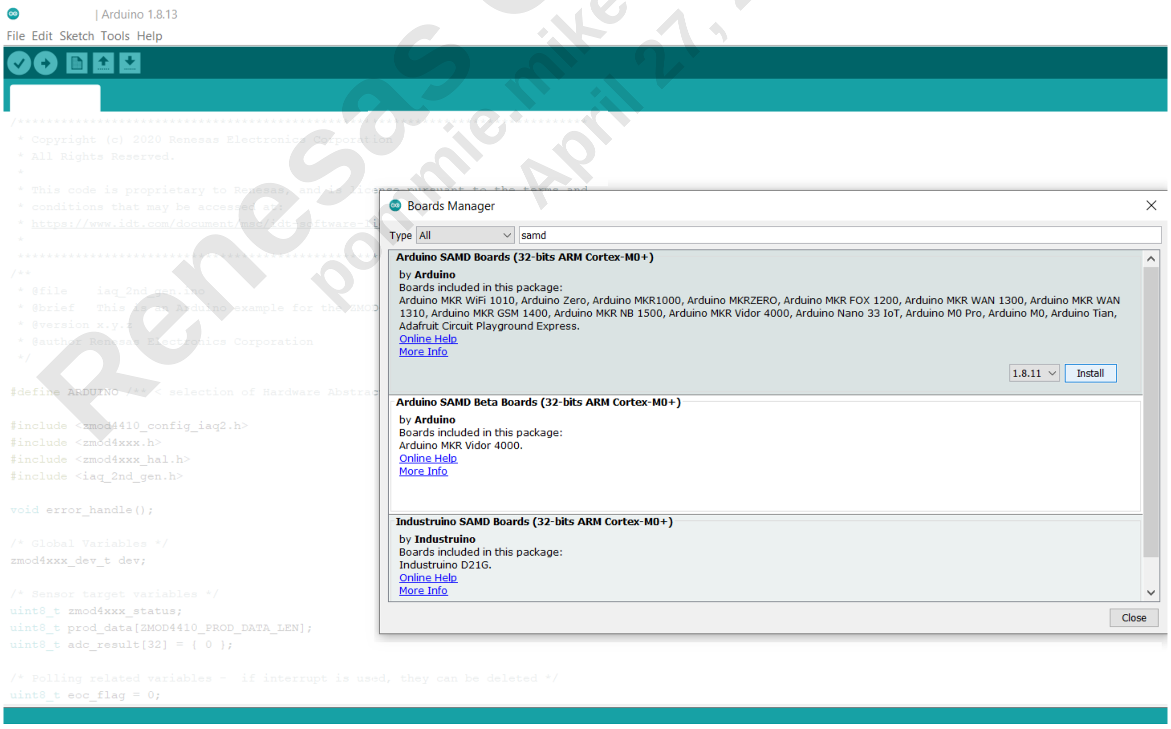
4. Select the Renesas_ZMOD4410- xxx_ Example_Arduino.zip file.

5. Select “File > Examples > Corresponding examples (Renesas_ZMOD4410_ xxx _Example_Arduino).” A new Arduino IDE window opens automatically with examples main file.



6. Install the “Arduino SAMD (32-bit ARM Cortex-M0+)” Boards library under “Tools > Board > Board Manager”. If it already exists, skip this step. Type “Arduino SAMD Boards” in the search field and click “Install” button in “Arduino SAMD (32-bit ARM Cortex-M0+)” field. This step is not required for AVR ATmega32 based boards because the AVR Boards library is installed by default.





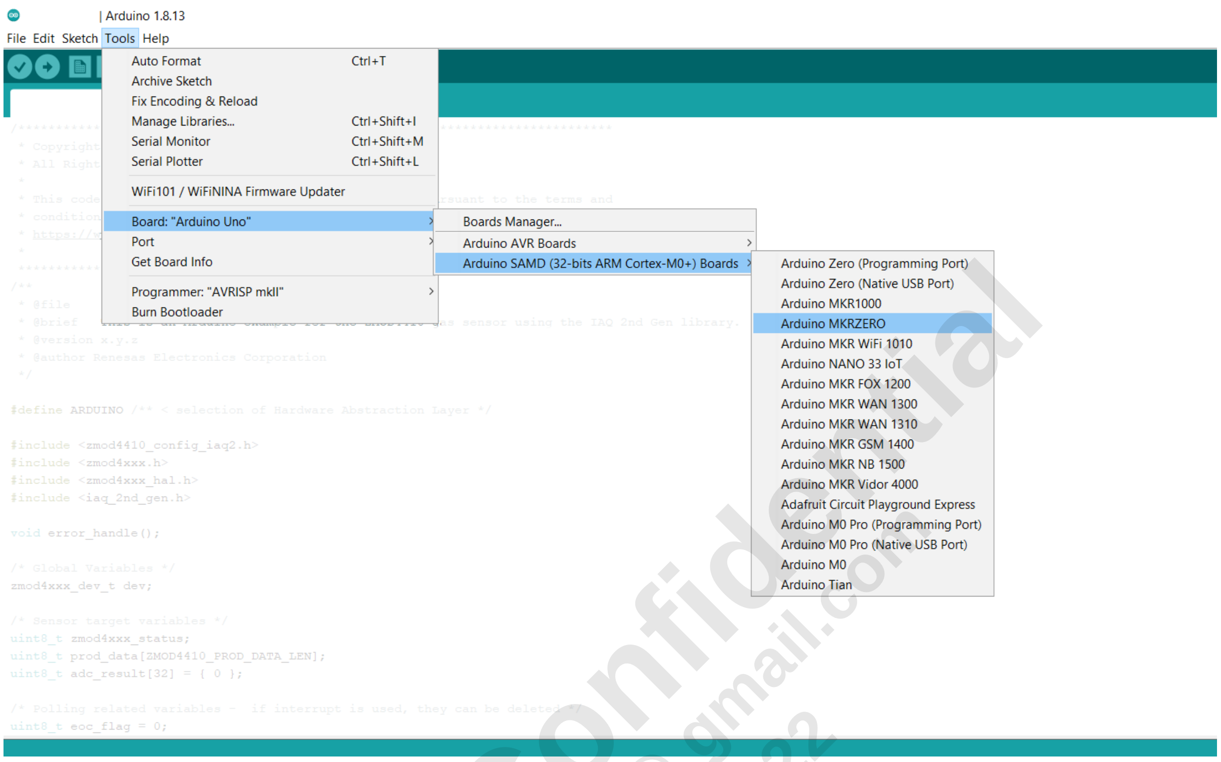
```
/*******************************************************************
 * Copyright (c) 2020 Renesas Electronics Corporation
 * All Rights Reserved.
 *
 * This code is proprietary to Renesas, and is license pursuant to the terms and
 * conditions that may be accessed at:
 * https://www.idt.com/document/msc/idt-software-li
 *
 *******************************************************
/**
 * @file    iaq_2nd_gen.ino
 * @brief   This is an Arduino example for the ZMOD
 * @version x.y.z
 * @author Renesas Electronics Corporation
 */

#define ARDUINO /**< selection of Hardware Abstrac

#include <zmod4410_config_iaq2.h>
#include <zmod4xxx.h>
#include <zmod4xxx_hal.h>
#include <iaq_2nd_gen.h>

void error_handle();

/* Global Variables */
zmod4xxx_dev_t dev;

/* Sensor target variables */
uint8_t zmod4xxx_status;
uint8_t prod_data[ZMOD4410_PROD_DATA_LEN];
uint8_t adc_result[32] = { 0 };

/* Polling related variables - if interrupt is used, they can be deleted */
uint8_t eoc_flag = 0;
```

Arduino SAMD Boards (32-bits ARM Cortex-M0+)
by **Arduino**
Boards included in this package:
Arduino MKR WiFi 1010, Arduino Zero, Arduino MKR1000, Arduino MKRZERO, Arduino MKR FOX 1200, Arduino MKR WAN 1300, Arduino MKR WAN 1310, Arduino MKR GSM 1400, Arduino MKR NB 1500, Arduino MKR Vidor 4000, Arduino Nano 33 IoT, Arduino M0 Pro, Arduino M0, Arduino Tian, Adafruit Circuit Playground Express.
Online Help
More Info

Arduino SAMD Beta Boards (32-bits ARM Cortex-M0+)
by **Arduino**
Boards included in this package:
Arduino MKR Vidor 4000.
Online Help
More Info

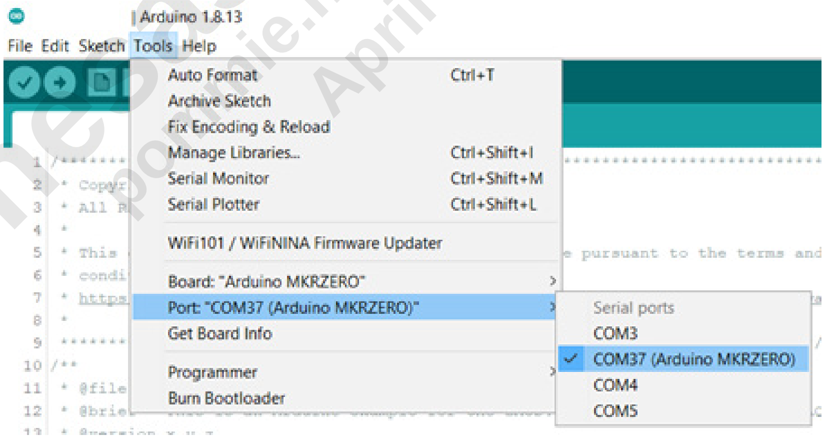
Industruino SAMD Boards (32-bits ARM Cortex-M0+)
by **Industruino**
Boards included in this package:
Industruino D21G.
Online Help
More Info

7. Select the target board with e.g. “Tools->Board > Arduino SAMD (32-bits ARM Cortex-M0+) > Arduino MKRZERO”

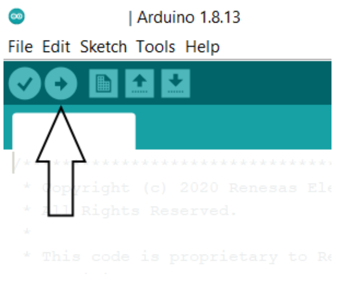

| Arduino 1.8.13
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Boards Manager...
Arduino AVR Boards
Arduino SAMD (32-bits ARM Cortex-M0+) Boards
Arduino Zero (Programming Port)
Arduino Zero (Native USB Port)
Arduino MKR1000
Arduino MKRZERO
Arduino MKR WiFi 1010
Arduino NANO 33 IoT
Arduino MKR FOX 1200
Arduino MKR WAN 1300
Arduino MKR WAN 1310
Arduino MKR GSM 1400
Arduino MKR NB 1500
Arduino MKR Vidor 4000
Adafruit Circuit Playground Express
Arduino M0 Pro (Programming Port)
Arduino M0 Pro (Native USB Port)
Arduino M0
Arduino Tian

```
#define ARDUINO /** < selection of Hardware Abstraction Layer */

#include <zmod4410_config_iaq2.h>
#include <zmod4xxx.h>
#include <zmod4xxx_hal.h>
#include <iaq_2nd_gen.h>

void error_handle();

/* Global Variables */
zmod4xxx_dev_t dev;

/* Sensor target variables */
uint8_t zmod4xxx_status;
uint8_t prod_data[ZMOD4410_PROD_DATA_LEN];
uint8_t adc_result[32] = { 0 };

/* Polling related variables - if interrupt is used, they can be deleted */
uint8_t eoc_flag = 0;
```

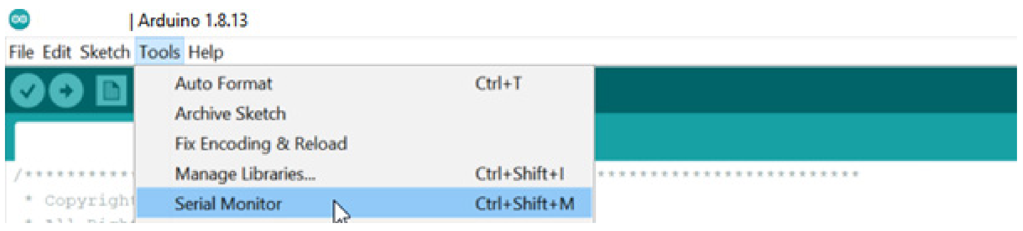
8. Compile the example with the “Verify” icon.
9. Select the connected port with “Tools -> Port -> (Connected Port)”. The correct COM-Port should show your Arduino board name.


| Arduino 1.8.13
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino MKRZERO"
Port: "COM37 (Arduino MKRZERO)"
Get Board Info
Programmer
Burn Bootloader
Serial ports
COM3
✓ COM37 (Arduino MKRZERO)
COM4
COM5

10. Load the program into target hardware with “Upload” icon.



11. Check results with the Serial Monitor (Tools -> Serial Monitor).


| Arduino 1.8.13
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M

4.6 Optional Library: Cleaning

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. Use *zmod4xxx_cleaning* library for this purpose. The example code shows how to use the cleaning function. The MOx resistance usually will 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. An alternative might be to consider the package option with assembly sticker (for more information, see Package Options in the *ZMOD4410 Datasheet*). The cleaning function is blocking the microcontroller.

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 so that it is not used by default.

5. Adapting the Programming Example for Target Hardware

5.1 System Hierarchy and Implementation Steps

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

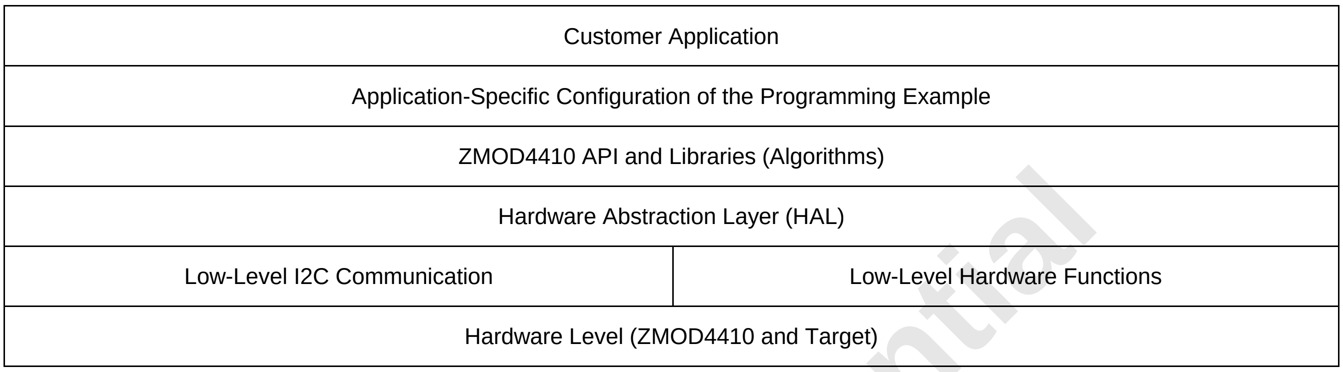

Customer Application
Application-Specific Configuration of the Programming Example
ZMOD4410 API and Libraries (Algorithms)
Hardware Abstraction Layer (HAL)
Low-Level I2C Communication
Low-Level Hardware Functions
Hardware Level (ZMOD4410 and Target)

Figure 2. System Hierarchy

The low-level I^{2}C 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 recommended:

1. Establish I^{2}C communication and conduct register test. 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 I^{2}C 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*. If an error occurs, check the following table for solutions. Note that ZMOD API cannot detect a wrong I2C implementation. Each error may occur also with a wrong I2C implementation.

Table 7. Error Codes (Cont. on Next Page)

Error Code	Error	Description	Solution
0	ZMOD4XXX_OK	No error.	
-1	ERROR_INIT_OUT_OF_RANGE	The initialization value is out of range.	Not used.
-2	ERROR_GAS_TIMEOUT	A previous measurement is running that could not be stopped or sensor does not respond.	1. Try to reset the sensor by powering it off/on or toggling the reset pin. Afterwards start the usual Program Flow as shown in section "Description of the Programming Examples". 2. Check your I2C wrapper functions for I2C read and write. Best is to analyze the voltage levels of the SDA/SCL line and check if they match the pattern described in figure "I2C Data Transmission Protocol" in Datasheet. Do a register check as requested in section "I2C Interface and Data Transmission Protocol" in Datasheet. Check also multiple register write and read out.
-3	ERROR_I2C	I2C communication was not successful.	1. 1.If available, check the error code of your parent I2C functions used in the ZMOD HAL for I2C_write/I2C_read implementation. 2. Check your I2C wrapper functions for I2C read and write. Best is to analyze the voltage levels of the SDA/SCL line and check if they match the pattern described in figure "I2C Data Transmission Protocol" in Datasheet. Do a register check as requested in section "I2C Interface and Data Transmission Protocol" in Datasheet. Check also multiple register write and read out.
-4	ERROR_SENSOR_UNSUPPORTED	The Firmware configuration used does not match the sensor module.	1. Check the part number of your device. Go to the product webpage www.renesas.com/zmod4410. Under the section "Downloads", you find the right firmware for ZMOD4410. Replace it. 2. Check your I2C wrapper functions for I2C read and write. Best is to analyze the voltage levels of the SDA/SCL line and check if they match the pattern described in figure "I2C Data Transmission Protocol" in Datasheet. Do a register check as requested in section "I2C Interface and Data Transmission Protocol" in Datasheet. Check also multiple register write and read out.
-5	ERROR_CONFIG_MISSING	There is no pointer to a valid configuration.	Not used.

Error Code	Error	Description	Solution
-6	ERROR_ACCESS_CONFLICT	Invalid ADC results due to a still running measurement while results readout.	1. Check if the delay function is correctly implemented. You can use a scope plot of a GPIO pin that is switched on and off. The delay function must introduce delays in milliseconds. 2. Check measurement timing by comparing your flow with the Program Flow as shown in section "Description of the Programming Examples". Figure 3 show graphically the right timing. Make sure to start a result readout after active measurement phase finished. See hints on measurement timing in section "Interrupt Usage and Measurement Timing". 3. Check your I2C wrapper functions for I2C read and write. Best is to analyze the voltage levels of the SDA/SCL line and check if they match the pattern described in figure "I2C Data Transmission Protocol" in Datasheet. Do a register check as requested in section "I2C Interface and Data Transmission Protocol" in Datasheet. Check also multiple register write and read out.
-7	ERROR_POR_EVENT	An unexpected reset of the sensor occurred.	1. Check stability of power supply and power lines for, e.g. cross-talk. After a Power-On reset the sensor lost its configuration and must be reconfigured. If the host-controller did not lose its memory due to a restart start with *zmod4xxx_prepare_sensor* function and continue in the Program flow as shown in section "Description of the Programming Examples".
-8	ERROR_CLEANING	The maximum numbers of cleaning cycles ran on this sensor. *zmod4xxx_cleaning_run* function has no effect anymore.	1. Using cleaning to often can harm the sensor module. The cleaning cannot be used anymore on this sensor module. Comment out the function *zmod4xxx_cleaning_run* if not needed.

-9	ERROR_NULL_PTR	The dev structure did not receive the pointers for I2C read, write and/or delay.	1. The init_hardware function (located in dependencies/zmod4xxx_api/HAL directory) contains assigning the variable of dev *read, *write and delay function pointers. These three I2C functions have to be generated for the corresponding hardware and assigned in the *init_hardware* function. This is exemplary shown in *hal_hicom.c*:
			`dev->read = hicom_i2c_read;`
			`dev->write = hicom_i2c_write;`
			`dev->delay_ms = hicom_sleep;`
			Check if the assignment was done.

5.3 Interrupt Usage and Measurement Timing

The programming examples are written in polling mode and with delays. The microcontroller is blocked during these time periods. Depending on target hardware and the application, this can be avoided by the use of interrupts. The whole measurement sequences for each example are displayed in the following figure.

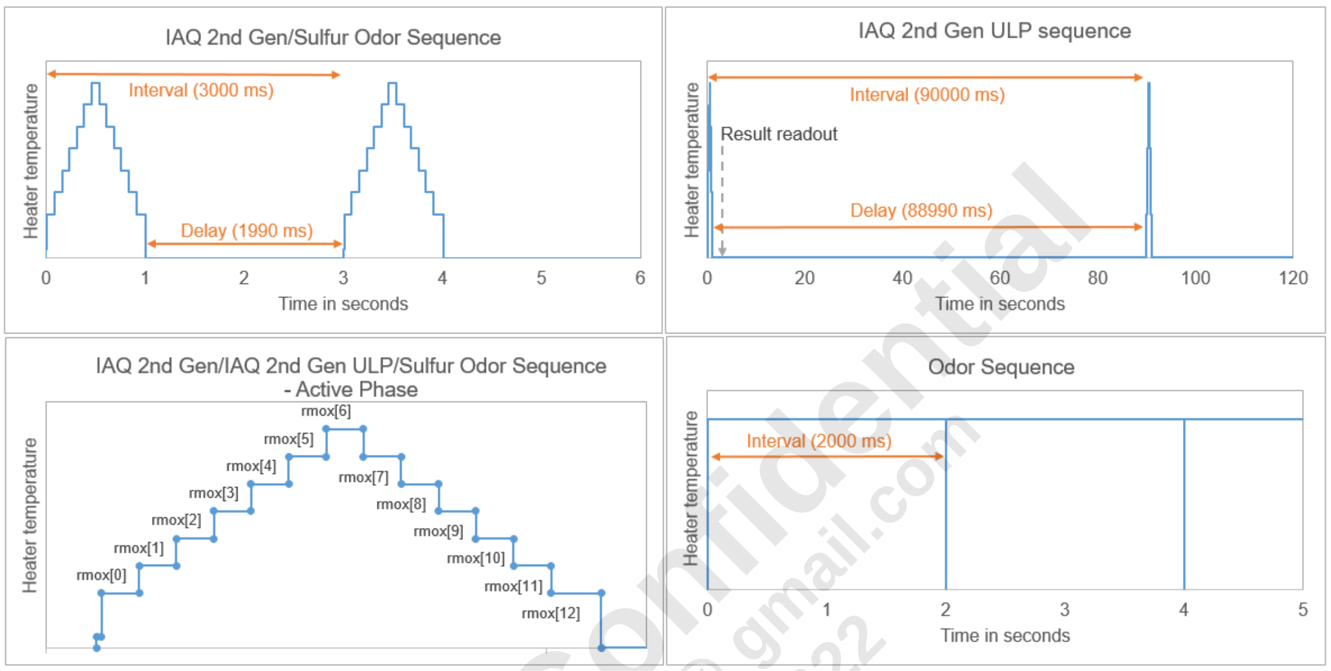


Figure 3. Measurement Sequences

An active measurement is indicated with a heater temperature greater zero. To lower power consumption, some modes have a delay afterwards. This timing must be kept exactly with a maximum deviation of 5%. With Odor Mode, the measurement is automatically restarted by the sensor itself. For the modes IAQ 2nd Gen, IAQ 2nd Gen ULP, and Sulfur Odor the measurement must be restarted with the API command *zmod4xxx_start_measurement*.

The following interrupt usages are possible:

- Using ZMODs Interrupt pin (INT) – This pin indicates the end of a measurement with a falling edge and stays LOW until the next measurement is restarted regardless if the results are read or not. The LOW phase after each measurement can be very short and an edge detection is needed.
  - Modes IAQ 2nd Gen, IAQ 2nd Gen ULP, and Sulfur Odor: Additionally an interrupt timer to start each measurement with the measurement interval (see Figure 3) is needed.
- Using Timer-based interrupts – Some target hardware may use timer-based interrupts. As an alternative to using the INT pin, a timer interrupt can be used to wait until the end of the active measurement phase. Note that an ADC read-out just before the end of the active measurement phase when results are written to the registers will lead to an error. When replacing the polling loop or delay make sure that the measurement is completed before ADC readout by checking with API function *zmod4xxx_read_status* and compare the output variable *zmod4xxx_status* with *STATUS_SEQUENCER_RUNNING_MASK*. An AND link of both should give zero, otherwise the measurement was not completed.
  - Modes IAQ 2nd Gen, IAQ 2nd Gen ULP and Sulfur Odor: Additionally the measurement must be started periodically with the corresponding measurement intervals (see Figure 3). The timing deviation should be below 5%. Another option for these modes is to use just one timer interrupt with the measurement interval. Then, the ADC result read-out, error check, and algorithm calculation is done just before starting the next measurement.

5.4 Adaptions to Follow C90 Standard

ZMOD4410 firmware supports C99 standard and later. A few configuration 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* must 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 EVK Programming Examples are 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 must be installed. The folder structure should be identical to that in the download package. The procedure is described in the IAQ 2nd Gen Example (*iaq_2nd_gen*). To adapt it for the other example, replace the corresponding name (*iaq_2nd_gen_ulp, odor, sulfur_odor*).

1. Install MinGW:
   a. MinGW (32 bit) must be used. Mingw64 will not work due to the 32-bit FTDI library for the EVK HiCom board.
   b. Download *mingw-get-setup.exe* from https://osdn.net/projects/mingw/releases/.
   c. The downloaded executable file installs “Install MinGW Installation Manager”.
   d. Select required packages:
      i. mingw-developer-toolkit-bin
      ii. mingw32-base-bin
      iii. mingw32-gcc-g++-bin
      iv. msys-base-bin.
   e. Click “Installation” from the left-top corner and select “Update Catalogue”.
   f. Finish the installation.
2. Add the *mingw-gcc* in system path:
   a. Open “Control Panel”, select “System”, select “Advanced System Settings”, then select “Environment Variables”
   b. Find “Path” in System Variables then add *C:\MinGW\bin* (change the path in case MinGW is installed in different location).
3. Compiling:
   a. Go to Command Prompt and change to the following directory of the example folder:
      [...]\Renesas_ZMOD4410_IAQ_2nd_Gen_Example\ zmod4xxx_evk_example
   b. Execute the following command in one line:
      ```
      gcc src\*.c HAL\*.c -o zmod4410_iaq_2nd_gen_example_custom.exe -DHICOM -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!
   c. An executable file called *zmod4410_iaq_2nd_gen_example_custom.exe* will be created.

6. Revision History

Revision	Date	Description
1.09	Dec.2.21	▪ Added IAQ 2nd Gen ULP Operation Mode description ▪ Added error code description ▪ Removed IAQ 1st Gen descriptions (legacy) ▪ Completed other minor changes
1.08	Aug.19.21	▪ Added Arduino description and updated target and compiler list. ▪ Added stack RAM usage. ▪ Corrected and reworked Program Flows. ▪ Extended “Interrupt Usage” description. ▪ Completed other minor changes
1.07	Sep.24.20	▪ 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.
1.06	May.27.20	▪ Completed many changes throughout the document.
1.05	Nov.14.19	▪ Cleaning procedure added and explained. ▪ Figure for file overview updated.
1.04	Feb.12.19	▪ 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.
1.03	Dec.5.18	▪ Update for Low Power Operation. ▪ Minor edits.
1.02	Sep.27.18	▪ 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.
1.00	Jun.11.18	Initial release.

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/