

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 methods of operation:

- 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 document, read the *ZMOD4410 Datasheet* and corresponding documentation on the ZMOD4410 product page.

Contents

1. Introduction	1
2. Requirements on Hardware to Operate ZMOD4410	3
3. Structure of ZMOD4410 Firmware	5
4. Description of the Programming Examples	6
4.1 IAQ 1 st Gen Example for EVK	6
4.2 IAQ 2 nd Gen Example for EVK	7
4.3 Odor Example for EVK	8
4.4 Sulfur Odor Example for EVK	9
4.5 Arduino Examples	10
4.6 Optional Libraries	14
4.6.1. Cleaning	14
4.6.2. Code Size Reduction	14
5. Adapting the Programming Example for Target Hardware	15
5.1 System Hierarchy and Implementation Steps	15
5.2 Error Codes	15
5.3 Interrupt Usage	16
5.4 Adaptions to Follow C90 Standard	17
5.5 How to Compile for EVK Hardware	18
6. Revision History	19

Figures

Figure 1. File Overview for ZMOD4410 Firmware	5
Figure 2. System Hierarchy	15
Figure 3. Measurement Sequences	16

Tables

Table 1. Exemplary Memory Footprint of ZMOD4410 Implementation on a Renesas RL78-G13 MCU	3
Table 2. Targets and Compilers Supported by Default	4
Table 3. IAQ 1 st Gen Program Flow for Continuous and Low Power Mode	6
Table 4. IAQ 2 nd Gen Program Flow	7
Table 5. Odor Program Flow	8
Table 6. Sulfur Odor Program Flow	9

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), see Table 1
- 1kB RAM for ZMOD4410-related operations (see 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 Renesas RL78-G13 MCU

	IAQ 2 nd Gen	IAQ 1 st Gen	Odor	Sulfur Odor
Program flash usage in kB	13.2	10.9	8.7	9.7
RAM usage (required variables) in bytes	250	266	168	256
RAM usage (stack size for library functions, worst case) in bytes	524	304	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 are 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-gnueabi-hf-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 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 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, 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.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 operation methods of the ZMOD4410. The Arduino examples have a similar structure, but some other features that facilitate operation with the Arduino board (see Arduino Examples).

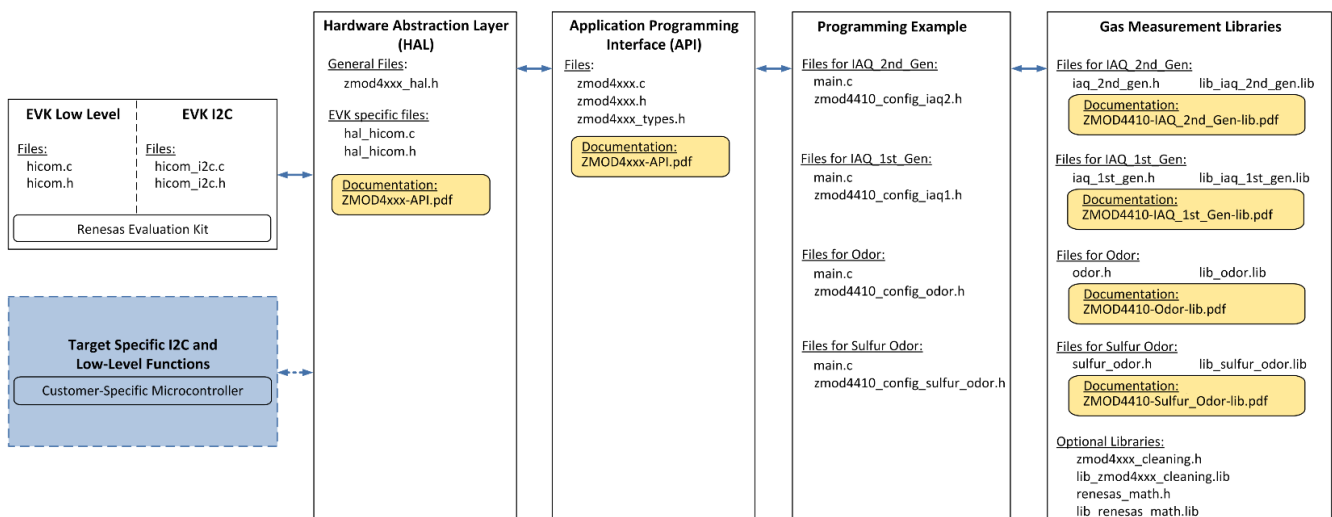


Figure 1. File Overview for ZMOD4410 Firmware

All files are part of zipped firmware packages available on the [ZMOD4410](#) product page under the Downloads section. Please 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 1st Gen Example for EVK

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 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 measurement 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.

Note: The blue colored lines in the following table can be run in an endless loop with polling or interrupt usage. Steps in brackets are only needed for Low Power Mode.

Table 3. IAQ 1st Gen Program Flow for Continuous and 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 product ID and configuration parameters.	This step is required to select the correct configuration for the sensor.	<code>zmod4xxx_read_sensor_info</code>
3	Calibration parameters are determined and measurement is configured.	This function must be called after every startup.	<code>zmod4xxx_prepare_sensor</code>
4	Initialize the IAQ (TVOC, eCO ₂) algorithm.	Gas Algorithm Library function. Initialize again after 48 hours of sensor operation.	<code>init_iaq_1st_gen</code>
5	Start the measurement.	Measurement runs in an endless loop for Standard Mode. Single Measurement is started in Low Power Mode.	<code>zmod4xxx_start_measurement</code>
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).	<code>zmod4xxx_read_status</code>
7	Read sensor ADC output.	Result contains raw sensor output.	<code>zmod4xxx_read_adc_result</code>
8	Algorithm calculation.	Calculate current MOx resistance R _{mox} , IAQ, TVOC, and eCO ₂ . First 15 samples (30 seconds) are used for sensor stabilization.	<code>calc_iaq_1st_gen</code>
(9)	Delay (5475ms).	This delay is necessary to maintain the right measurement timing and to call a measurement every 6 seconds with a maximum of 5% deviation.	-
(10)	Start next measurement.	One measurement is started.	<code>zmod4xxx_start_measurement</code>

4.2 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 nonvolatile 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, 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 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, eCO ₂) 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	Read sensor ADC output.	Result contains raw sensor output.	zmod4xxx_read_adc_result
8	Algorithm calculation.	Calculate current MOx resistance Rmox, IAQ, TVOC and eCO ₂ . First 60 samples (3 minutes) are used for sensor stabilization.	calc_iaq_2nd_gen
9	Delay (1990ms).	This delay is necessary to keep the right measurement timing and to call a measurement every 3 seconds with a maximum of 5% deviation.	-
10	Start next measurement.	One measurement is started.	zmod4xxx_start_measurement

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 nonvolatile 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.	<code>zmod4xxx_read_sensor_info</code>
3	Calibration parameters are determined and measurement is configured.	This function must be called after every startup.	<code>zmod4xxx_prepare_sensor</code>
5	Start the measurement.	Measurement runs in an endless loop.	<code>zmod4xxx_start_measurement</code>
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).	<code>zmod4xxx_read_status</code>
7	Read sensor ADC output.	Result contains raw sensor output.	<code>zmod4xxx_read_adc_result</code>
8	Get the MOx resistance value.	Get the MOx resistance value.	<code>zmod4xxx_calc_rmox</code>
9	Algorithm calculation.	Calculate Odor control state <code>cs_state</code> and Air Quality Change <code>conc_ratio</code> . First 15 samples (30 seconds) are used for sensor stabilization.	<code>calc_odor</code>

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 nonvolatile 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, 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 sensor stabilization.	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 of 5% deviation.	-
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:

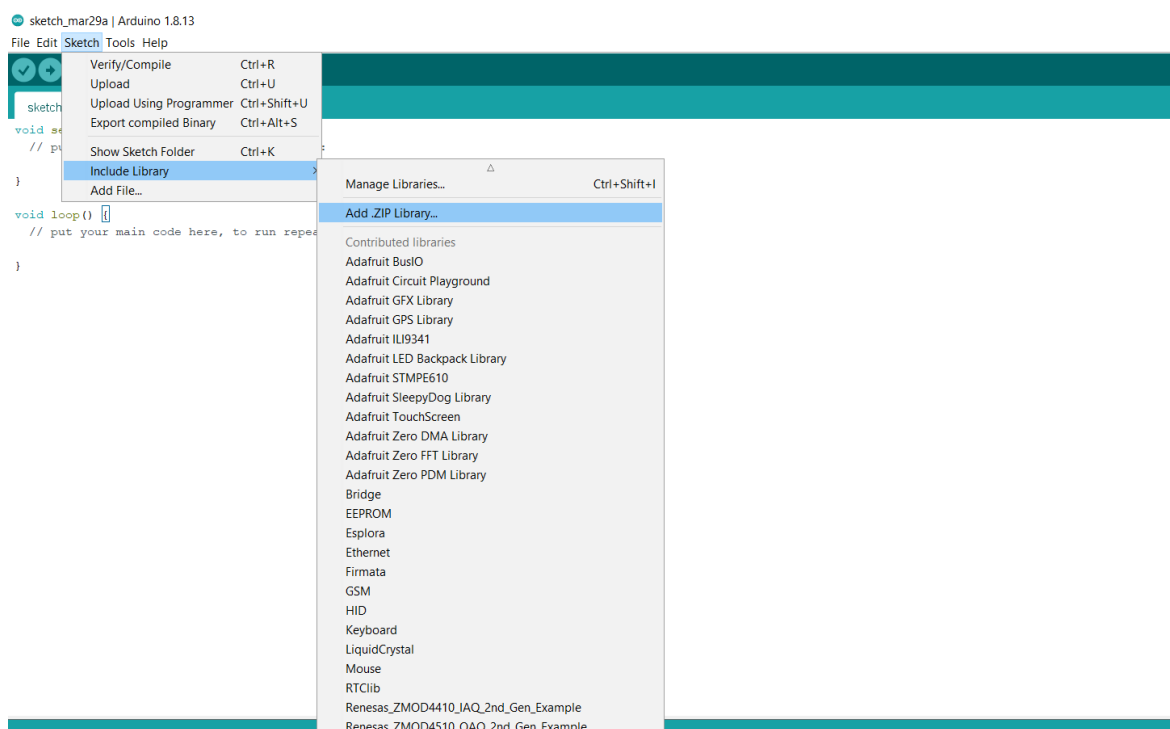
- Arduino Zero/MKR Zero
- Arduino MKR1000
- Arduino Nano 33 IoT
- Arduino M0
- etc.

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

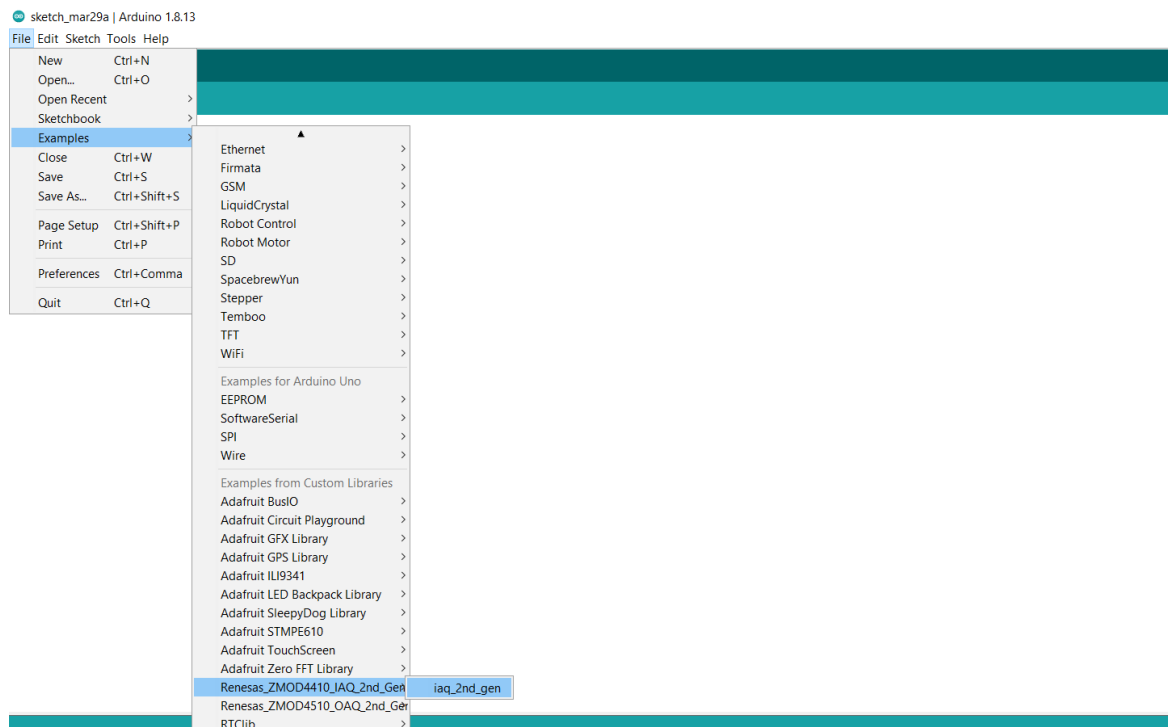
- Arduino Uno Rev3 (SMD)
- Arduino Nano
- Arduino Micro
- Arduino Leonardo
- etc.

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):

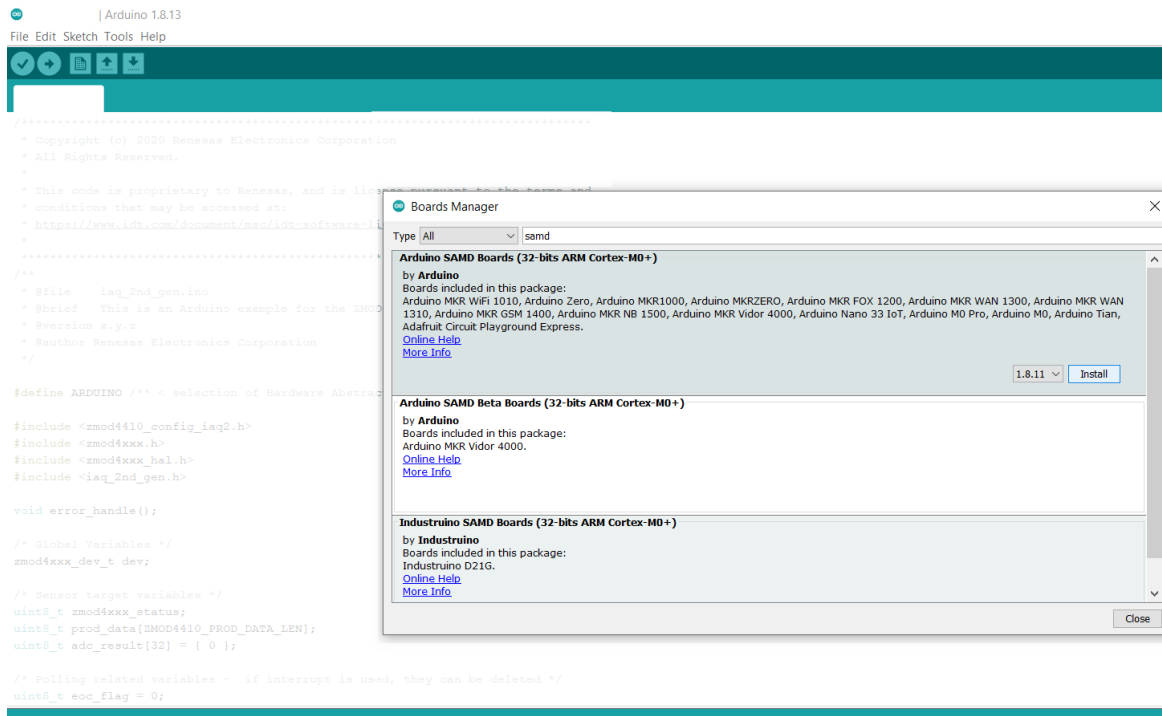
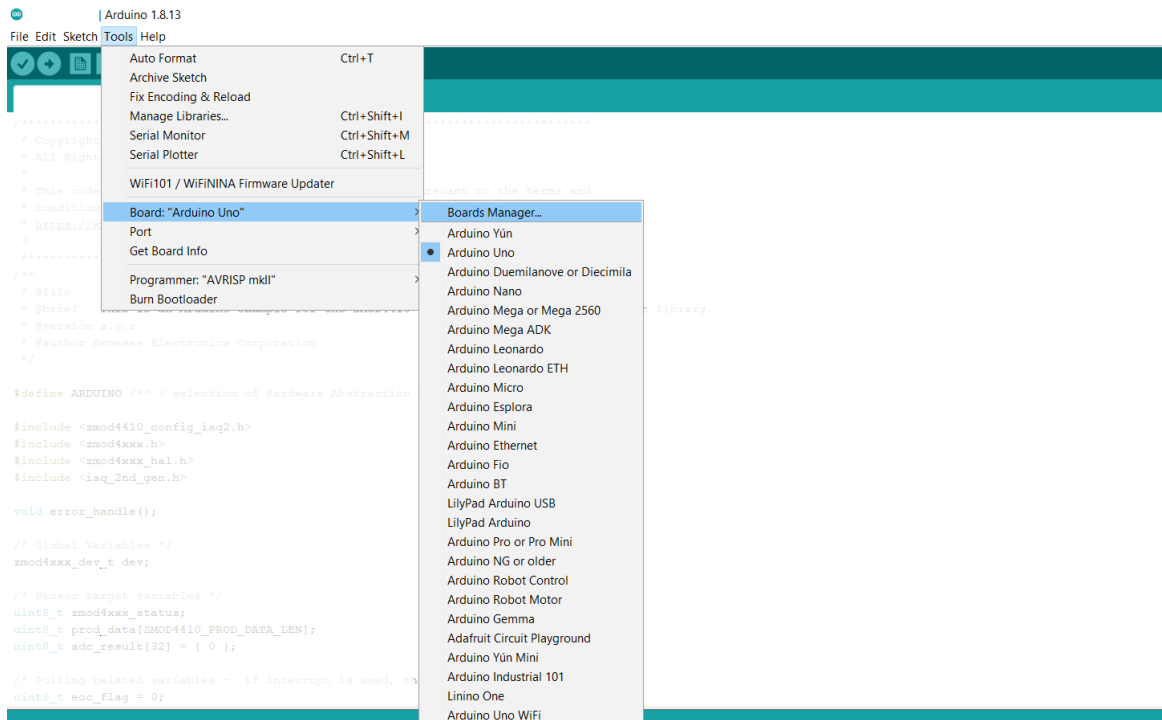
1. Connect the ZMOD4410 to the Arduino board. To connect the EVK Sensor Board, check the pin configuration on connector X1 in the *ZMOD4410 EVK User Manual* on the [ZMOD4410 EVK](#) product page.
2. Go to the Arduino example path (for example, [...]\Documents\Arduino\libraries) and check if a ZMOD4410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



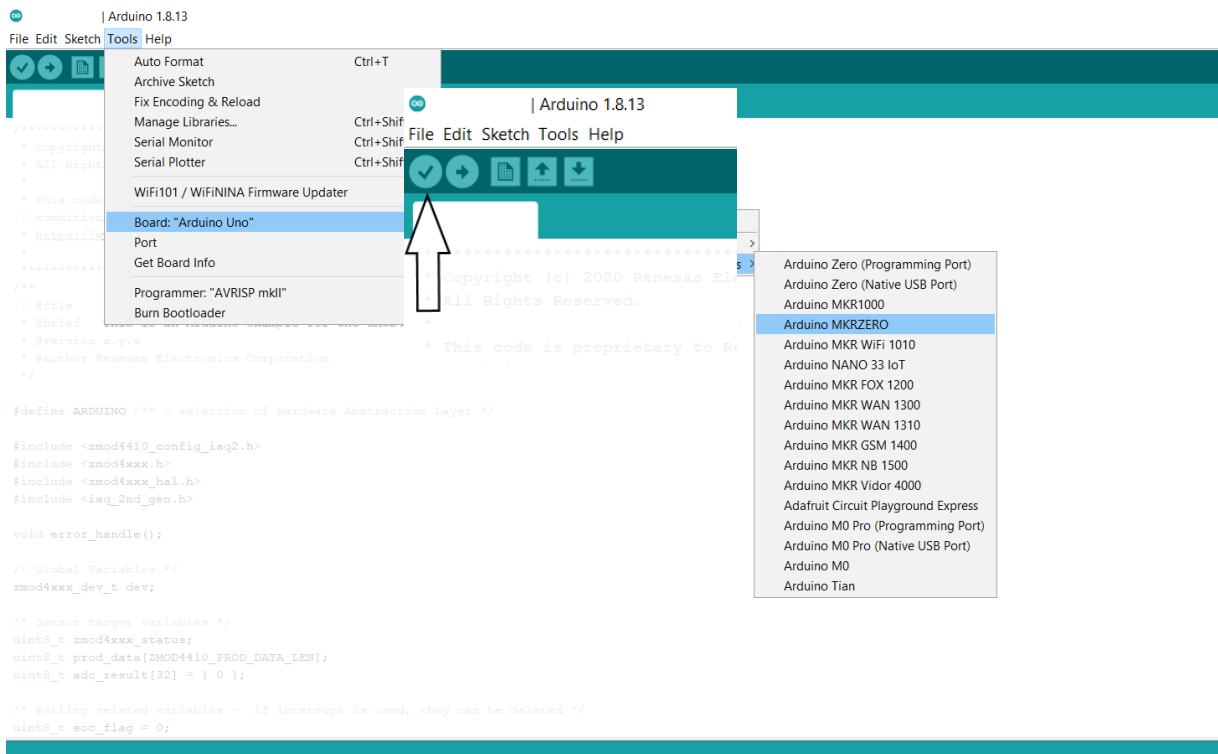
4. Select the Renesas_ZMOD4410- xxx_ Example_Arduino.zip file.
5. Select “File -> Examples -> Corresponding examples (Renesas_ZMOD4410_ xxx _Example_Arduino).” A new Arduino IDE window opens automatically with examples main file.



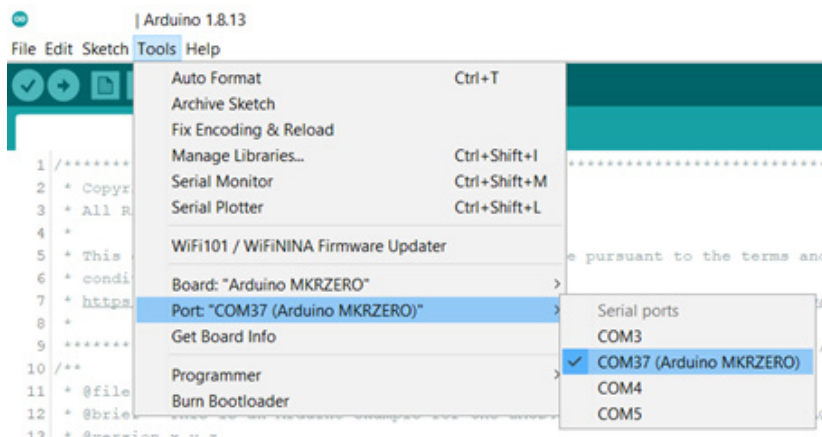
- Install the “Arduino SAMD (32-bit ARM Cortex-M0+)” Boards library under “Tools -> Board -> Board Manager”. If it already exists, skip this step. Type “Arduino SAMD Boards” in the search field and click “Install” button in “Arduino SAMD (32-bit ARM Cortex-M0+)” field. This step is not required for AVR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



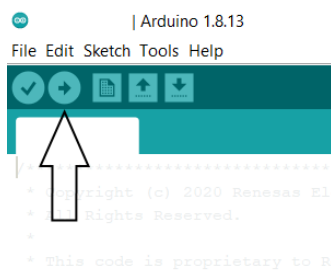
7. Select the target board with e.g. “Tools->Board->Arduino SAMD (32-bits ARM Cortex-M0+)->Arduino MKRZERO”



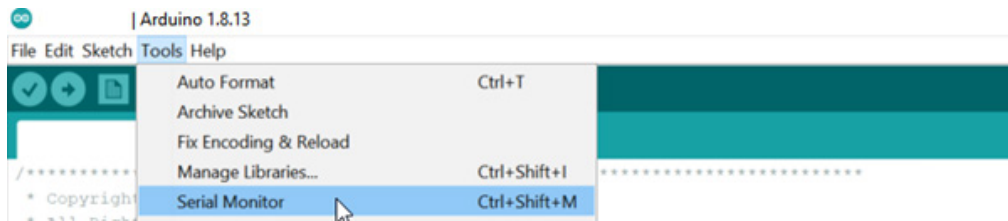
8. Compile the example with the “Verify” icon.
9. Select the connected port with “Tools -> Port -> (Connected Port)”. The correct COM-Port should show your Arduino board name.



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



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



4.6 Optional Libraries

4.6.1. 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.

4.6.2. 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. Check 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

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 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 recommended:

1. Establish I²C 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 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 the application it might be beneficial to make use of interrupts. 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. The LOW phase after each measurement can be very short and an edge detection is needed. The detection of an interrupt can replace `zmod4xxx_read_status()` calls with corresponding polling loops. These code sections contain a comment in the example code to point to the possibility of interrupt pin usage.
- Timer-based interrupts – Some target hardware may use timer-based interrupts. For this usage, the main measurement loop must be called periodically with the corresponding measurement intervals. The timing deviation should be below 5%. This procedure can replace the delays, which are used to ensure measurement timing, and the polling loops. An ADC read-out at the end of the active measurement phase when results are written to the registers will lead to an error. When replacing the polling loops make sure that measurement is completed. The measurement sequences for each example are shown in Figure 3.

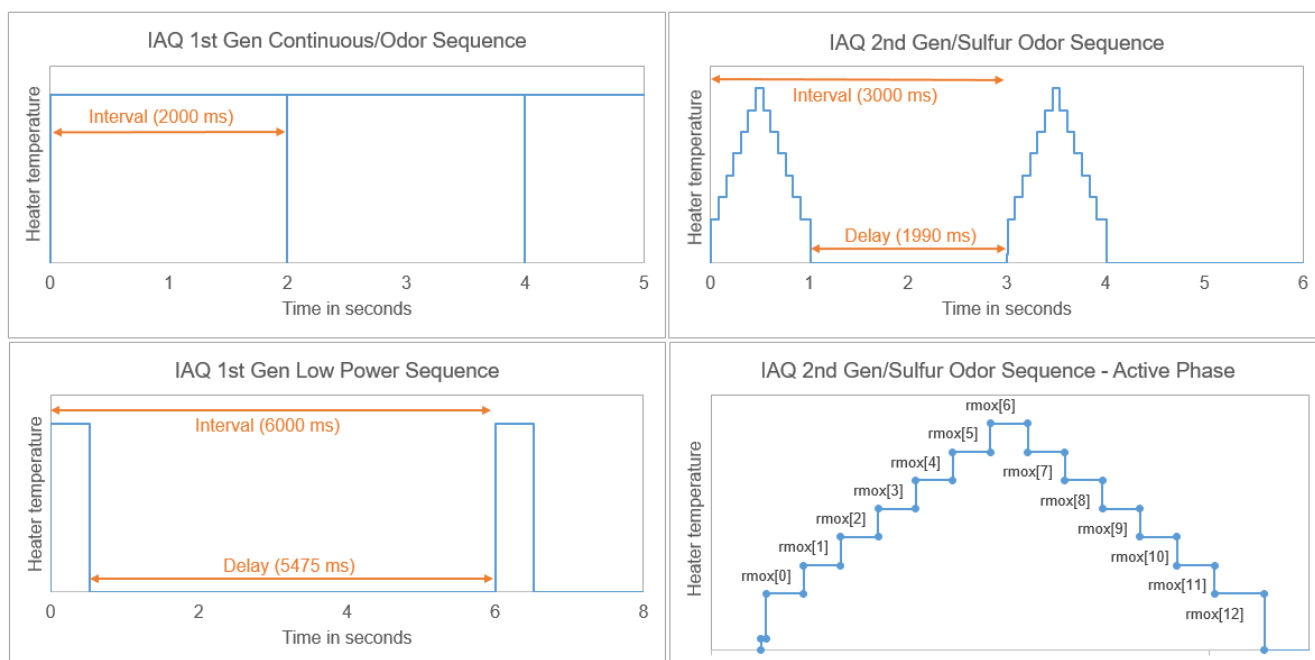


Figure 3. Measurement Sequences

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_1st_gen*, *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 -I..\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
1.08	Aug.19.21	<ul style="list-style-type: none">▪ 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	<ul style="list-style-type: none">▪ 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	<ul style="list-style-type: none">▪ Completed many changes throughout the document.
1.05	Nov.14.19	<ul style="list-style-type: none">▪ Cleaning procedure added and explained.▪ Figure for file overview updated.
1.04	Feb.12.19	<ul style="list-style-type: none">▪ 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	<ul style="list-style-type: none">▪ Update for Low Power Operation.▪ Minor edits.
1.02	Sep.27.18	<ul style="list-style-type: none">▪ 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.