ZMOD4510

1. Introduction

The ZMOD4510 Gas Sensor Module is highly configurable to meet various application needs. This document describes the general program flow to set up ZMOD4510 Gas Sensor Modules for gas measurements in a customer environment. The corresponding firmware package is provided on the Renesas ZMOD4510 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 ZMOD4510 evaluation kit (EVK) and with Arduino hardware. For instructions on assembly, connection, and installation of the EVK hardware and software, see the document titled *ZMOD4510 Evaluation Kit User Manual* on the ZMOD4510 EVK product page.

The ZMOD4510 has two methods of operation:

- OAQ 1st Gen – The OAQ algorithm ("oaq_1st_gen") outputs an Air Quality Index (AQI) based on the rating of the US Environmental Protection Agency (EPA)[1] gained on traditional gas sensor algorithms with a sampling rate of one minute.
- OAQ 2nd Gen – The embedded artificial intelligence (AI) algorithm ("oaq_2nd_gen") derived from machine learning outputs an Air Quality Index for ozone based on the rating of the US EPA and an ozone concentration with a sampling rate of two seconds. This method of operation offers a lower power consumption while keeping accurate and consistent sensor readings.

Recommendation: Before using this document, please read the *ZMOD4510 Datasheet* and corresponding documentation on the ZMOD4510 product page.

[1] Source: *AirNow*, United States Environmental Protection Agency, Office of Air Quality Planning and Standards (OAQPS), 2019

Contents

Figures

Tables

2. Requirements on Hardware to Operate ZMOD4510

To operate the ZMOD4510, 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 ZMOD4510-related firmware code (MCU architecture and compiler dependent), see Table 1
- 1kB RAM for ZMOD4510-related operations (see Table 1)
- Capability to perform I^2C 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 ZMOD4510 Implementation on a Renesas RL78-G13 MCU

	OAQ 2nd Gen	OAQ 1st Gen
Program flash usage in kB	12.9	10.2
RAM usage (required variables) in bytes	250	266
RAM usage (stack size for library functions, worst case) in bytes	544	244

The ZMOD4510 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 the following table.

Table 2. Targets and Compilers Supported by Default

Target	Compiler
Arduino SAMD Cortex-M0+	arm-none-eabi-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

Target	Compiler
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, please contact Renesas Technical Support.

3. Structure of ZMOD4510 Firmware

To operate the ZMOD4510 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 ZMOD4510 and a delay function. If the Renesas EVK is used, files for the EVK HiCom Communication Board are provided with the ZMOD4510 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 ZMOD4510 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 ZMOD4510. 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 ZMOD4510, perform measurements, display the data output for each specific example, and start the optional cleaning function. Each example contains one configuration file (*zmod4510_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 Air Quality Index (AQI). These algorithms cannot be used in parallel. This block also contains the optional cleaning library. The libraries are described in more detail in the documents *ZMOD4510-OAQ_1st_Gen-lib.pdf* and *ZMOD4510-OAQ_2nd_Gen-lib.pdf*.

To avoid naming conflicts, all API function names start with the prefix “`zmod4xxx`” in the ZMOD4510 code. This naming applies to all operation methods of the ZMOD4510. Arduino examples have a similar structure but some other features to easily operate with the Arduino board (see Arduino Examples).

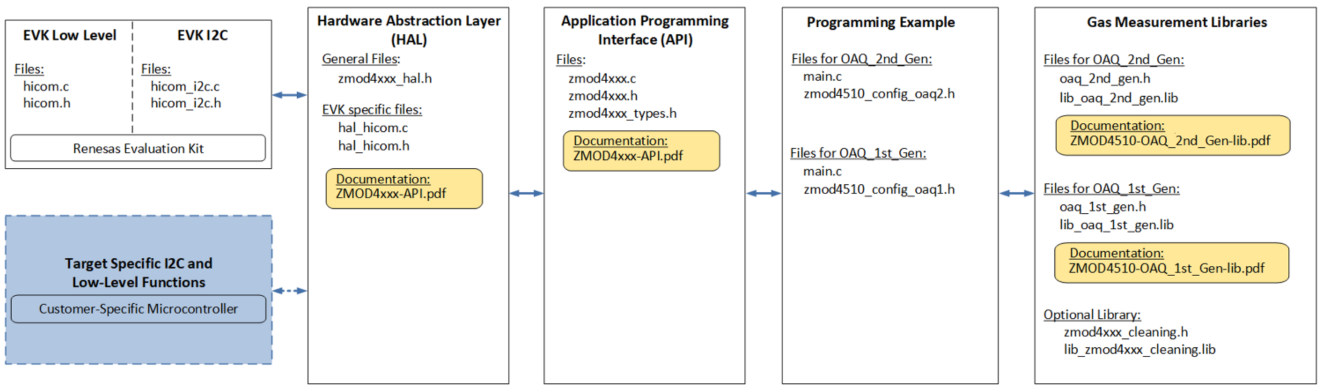


Figure 1. File Overview for ZMOD4510 Firmware

All files are part of zipped firmware packages available on the ZMOD4510 product page under the Downloads section.

4. Description of the Programming Examples

This section describes the programming examples and how to use the ZMOD4510 Gas Sensor Module. In the examples, the ZMOD4510 is initialized, the measurement is started, and measured values are outputted. The examples OAQ 1st Gen and OAQ 2nd Gen 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 *zmod4510_xxx_example.exe,* which are included in the firmware packages. Arduino examples will work for the described Arduino hardware (see Arduino Examples).

4.1 OAQ 1st Gen Example for EVK

The *main.c* file of the example contains the main program flow. First, the target-specific initializations are performed. The ZMOD4510 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 ZMOD4510 and reads its data. The raw data is subsequently processed and AQI algorithm result is calculated. The value is 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. OAQ 1st 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 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 OAQ algorithm.	Gas Algorithm Library function.	init_oaq_1st_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	Start next measurement.	Measurement needs to be restarted immediately after ADC readout to keep correct timing. It will run for 60 seconds.	zmod4xxx_start_measurement
9	Algorithm calculation.	Calculate current MOx resistance Rmox and 1h AQI rating according to EPA standard[2]. First 60 samples used for sensor stabilization.	calc_oaq_1st_gen

[2] Source: https://www.airnow.gov/sites/default/files/2020-05/aqi-technical-assistance-document-sept2018.pdf

4.2 OAQ 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 ZMOD4510 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 ZMOD4510 and reads its data. The raw data is subsequently processed, and the ozone gas concentration, fast output for AQI and standardized AQI 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 details, refer to the example code.

Table 4. OAQ 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 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 OAQ algorithm.	Gas Algorithm Library function.	init_oaq_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, ozone gas concentration in ppb, fast output for AQI and 1h/8h AQI rating according to EPA standard[3]. Relative humidity (in % RH) and temperature values (in °C) need to be provided. First 901 samples used for sensor stabilization.	calc_oaq_2nd_gen
9	Delay (1980ms).	This delay is necessary to keep the right measurement timing and call a measurement every 1.998 seconds with a maximum of 5% deviation.	-
10	Start next measurement.	One measurement is started.	zmod4xxx_start_measurement

[3] Source: https://www.airnow.gov/sites/default/files/2020-05/aqi-technical-assistance-document-sept2018.pdf

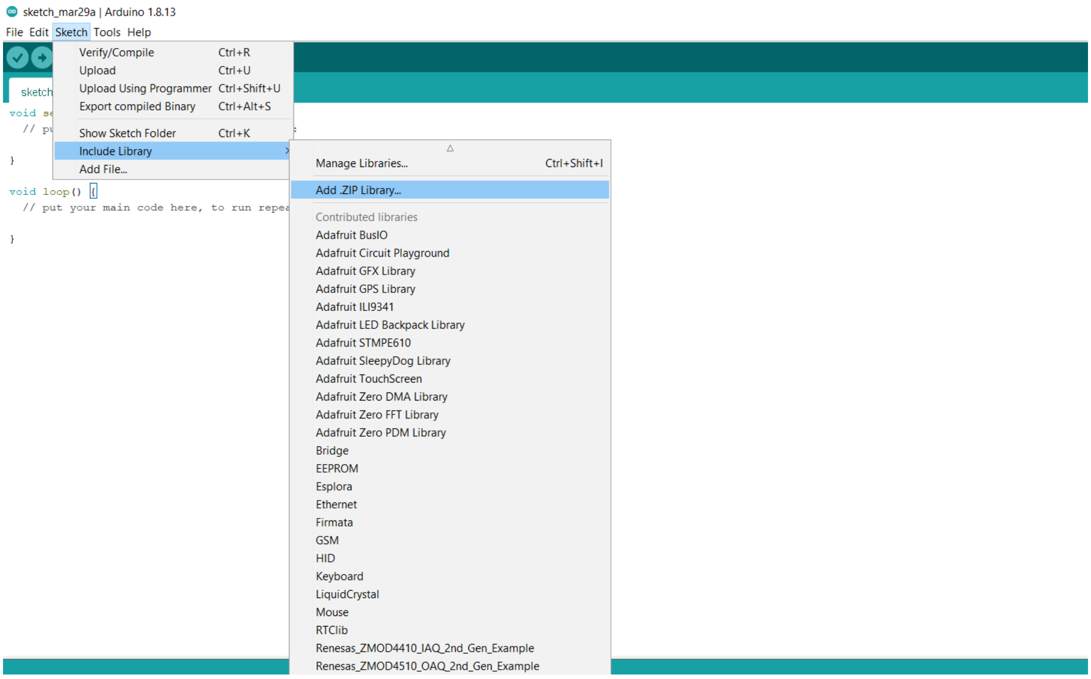
4.3 Arduino Examples

To setup a firmware for an Arduino target Renesas provides the two 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. The example supports SAMD 32-bits ARM Cortex-M0+ based Arduino-Hardware included in the SAMD Boards library, e.g.:

- Arduino Zero/MKR ZERO
- Arduino MKR1000
- Arduino NANO 33 IoT
- Arduino M0
- etc.

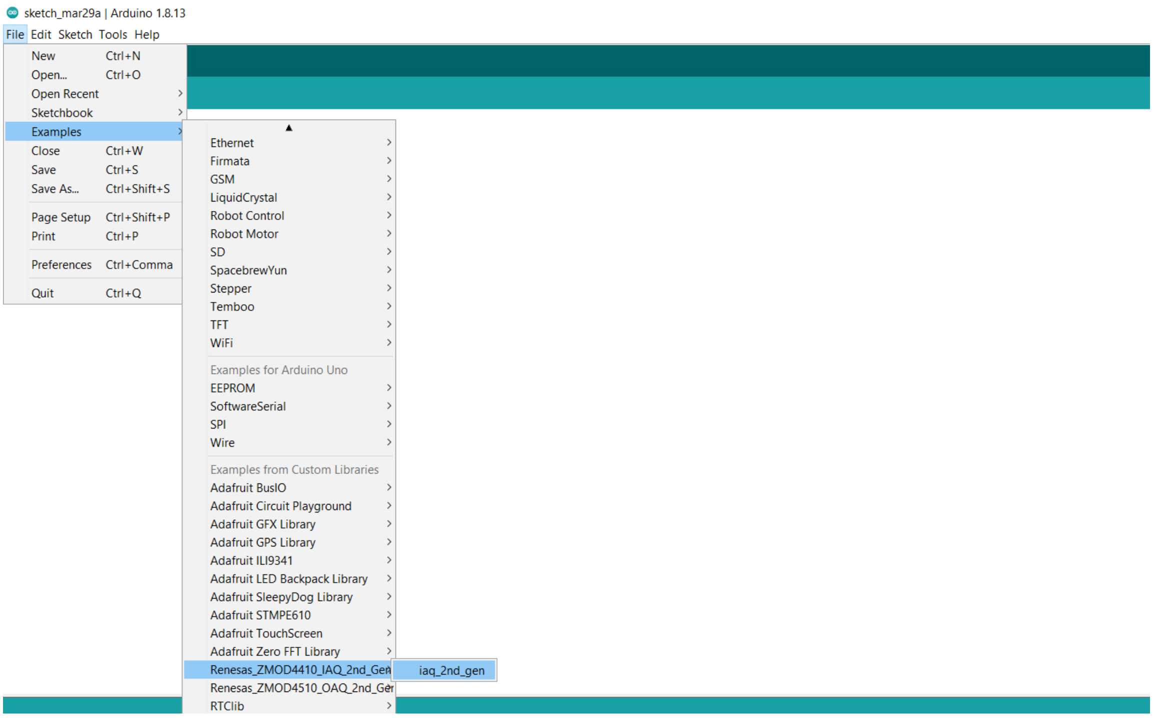
The Program Flows correspond to those depicted in the according EVK examples in OAQ 1st Gen Example for EVK and OAQ 2nd Gen Example for EVK. To get the Arduino example started, complete the following these steps:

1. Connect the ZMOD4510 to the Arduino board. To connect the EVK Sensor Board, check the pin configuration on connector X1 in the *ZMOD4510 EVK User Manual* on ZMOD4510 EVK product page.
2. Go to the Arduino example path (e.g. […]\Documents\Arduino\libraries) and check if a ZMOD4510 example is existing. Old example folders must be deleted.
3. Open Arduino IDE. Select "Sketch->Include Library->Add .ZIP library".

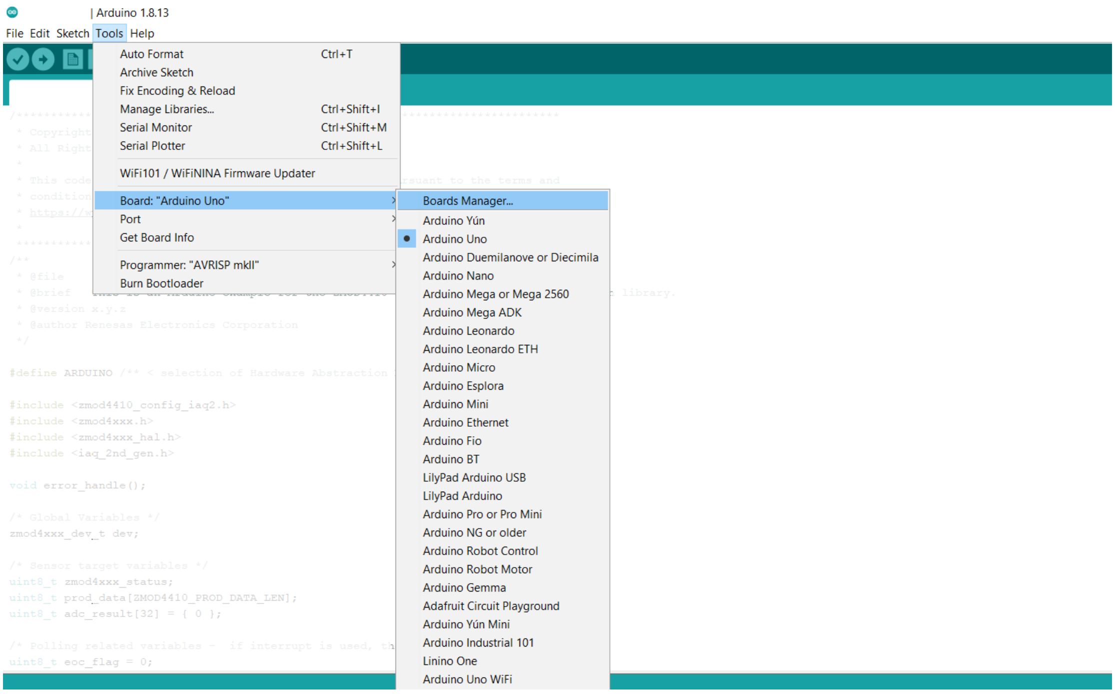


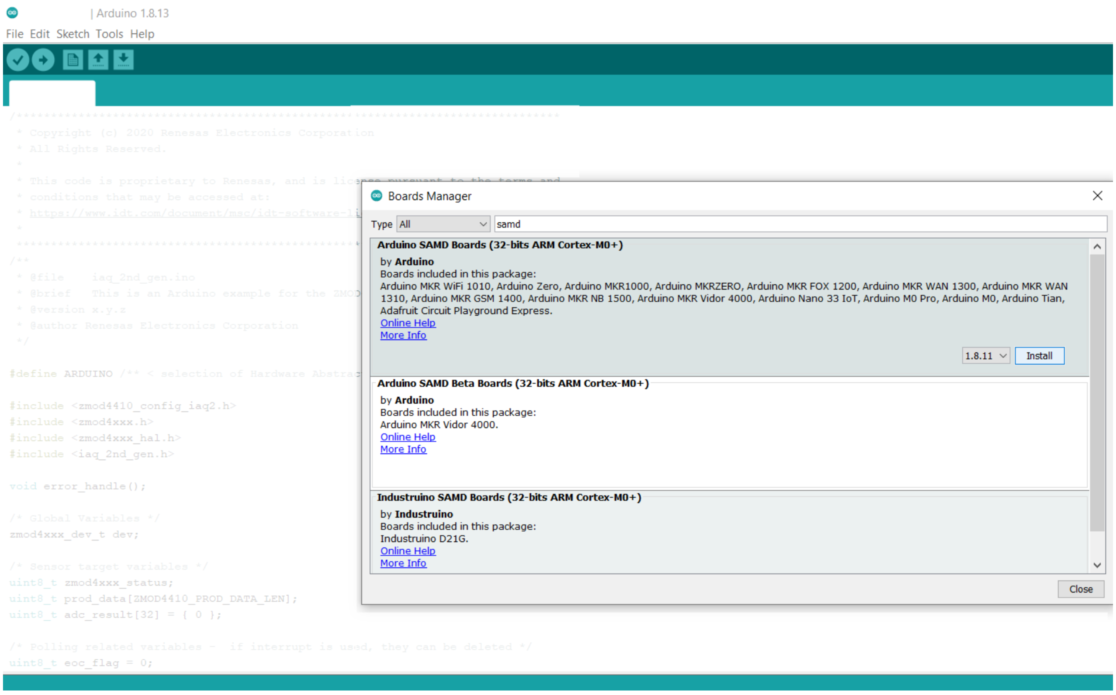
4. Select the Renesas_ZMOD4510-OAQ_xxx_Gen_Example_Arduino.zip file.

5. Select "File -> Examples -> Corresponding examples (Renesas_ZMOD4510_OAQ_xxx_Gen_Example_Arduino)." New Arduino IDE window opens automatically with examples main file.

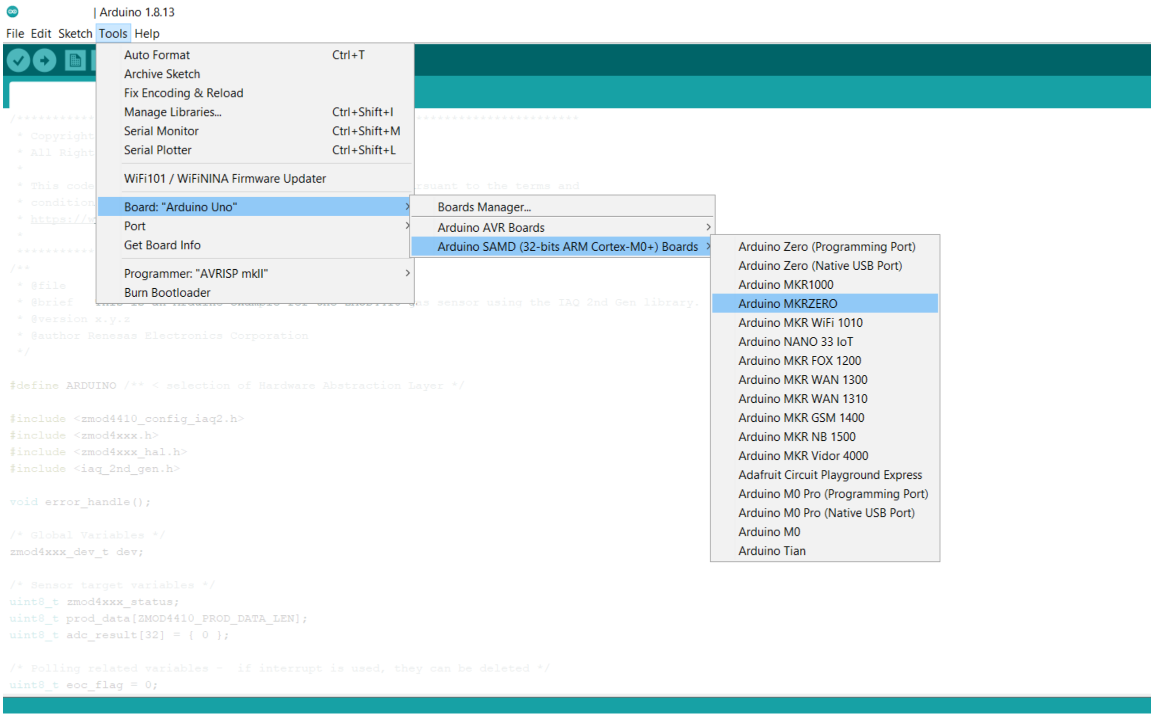


6. Install “Arduino SAMD (32-bits ARM Cortex-M0+)” Boards library under “Tools -> Board -> Board Manager”. If it already exists, you can skip this step. Type “Arduino SAMD Boards” in search field and click “Install” button in “Arduino SAMD (32-bits ARM Cortex-M0+)” field.

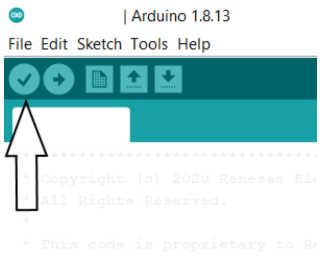




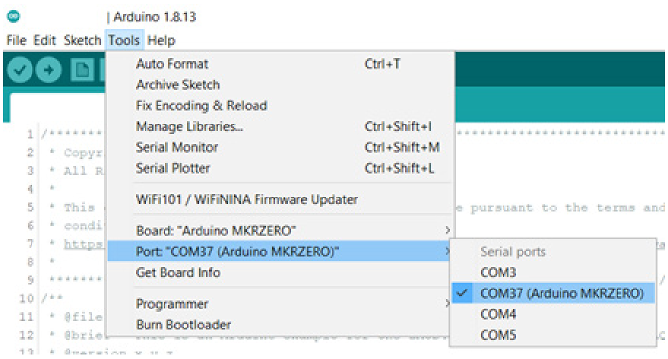
7. Select the target board with e.g. “Tools->Board->Arduino SAMD (32-bits ARM Cortex-M0+)->Arduino MKRZERO”



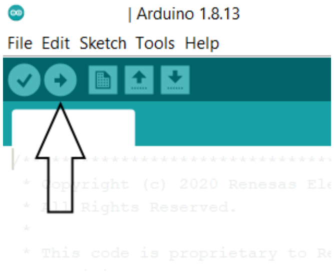
8. Compile the example with the “Verify” icon.



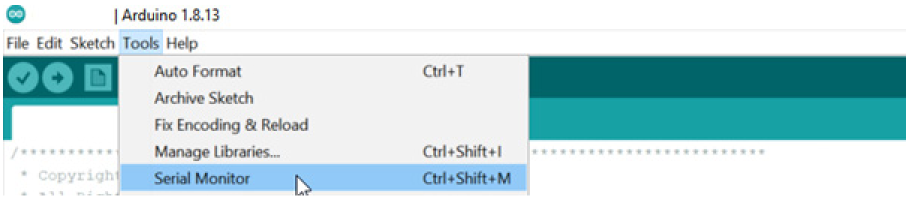
9. Select the connected port with “Tools -> Port -> (Connected Port)”. The correct COM-Port should show your Arduinos board name.



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



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



4.4 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. 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. An alternative might be to consider the package option with assembly sticker (for more information, see Package Options in the *ZMOD4510 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 to not use it by default.

5. Adapting the Programming Example for Target Hardware

5.1 System Hierarchy and Implementation Steps

The Renesas ZMOD4510 C API is located between the application and the hardware level.

Figure 2. System Hierarchy

Customer Application	
Application-Specific Configuration of the Programming Example	
ZMOD4510 API and Libraries (Algorithms)	
Hardware Abstraction Layer (HAL)	
Low-Level I2C Communication	Low-Level Hardware Functions
Hardware Level (ZMOD4510 and Target)	

The low-level I^2C 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^2C communication and conduct register test. Please find detailed hints in the "I2C Interface and Data Transmission Protocol" section of the *ZMOD4510 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^2C 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 ZMOD4510's interrupt pin (INT) is not used. However, depending on target hardware and 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. Care must be taken for OAQ 1st Gen. In this mode, there is no sleep delay between the measurements. The LOW phase after each measurement should be very short and an edge detection may be useful. The detection of an interrupt can replace *zmod4xxx_read_status()* calls and 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 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 and polling loops. The measurement intervals for each example are as follows.
  - OAQ 2nd Gen – 1.998 seconds (replace the delay of 1980ms)
  - OAQ 1st Gen – 1 minute (no delays used).

5.4 Adaptions to Follow C90 Standard

ZMOD4510 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 *zmod4510_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 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 on the OAQ 2nd Gen Example (*oaq_2nd_gen*). To adapt it for the other example just replace the corresponding name (*oaq_1st_gen*).

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 installation.
2. Add the *mingw-gcc* in system path:
   a. Open “Control Panel”, select “System”, select “Advanced System Settings”, 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_ZMOD4510_OAQ_2nd_Gen_Example\ zmod4xxx_evk_example
   b. Execute the following command in one line:
   gcc src*.c HAL*.c -o zmod4510_oaq_2nd_gen_example_custom.exe -DHICOM -Isrc -IHAL
   -I..\gas-algorithm-libraries\oaq_2nd_gen\Windows\x86\mingw32 -L. -l:HAL\RSRFTCI2C.lib
   -l:..\gas-algorithm-libraries\oaq_2nd_gen\Windows\x86\mingw32\lib_oaq_2nd_gen.lib
   Note, gcc command may need admin rights!
   c. An executable file called *zmod4510_oaq_2nd_gen_example_custom.exe* will be created.

6. Revision History

Revision	Date	Description
1.1	Apr.30.21	▪ Added Operation Mode OAQ 2nd Gen and OAQ 1st Gen (old algorithm version) ▪ Added and explained Cleaning procedure ▪ Added and explained Arduino example ▪ Completed other minor improvements
1.0	Sep.4.19	Initial release.