XENSIV™ PAS CO2 Sensor Arduino Library

Infineon Technologies AG

May 15, 2023

CONTENT

1 Getting Started 3

2 Hardware Platforms 9

3 Library Installation 15

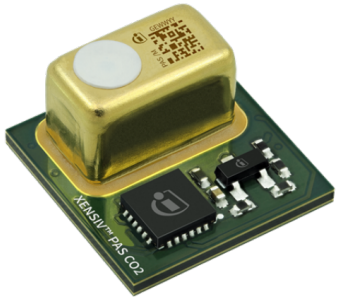
4 Examples 17

5 API Reference 19

6 License 27

Index 29

Welcome to the Infineon Photoacoustic Spectroscopy XENSIV™ PAS CO2 Sensor Arduino library docs!



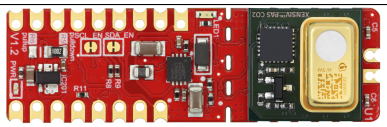
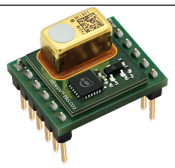
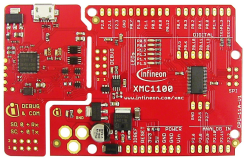
CHAPTER

ONE

GETTING STARTED

In this quick tutorial we will go through one of the XENSIV™ PAS CO2 sensor examples available using the Shield2Go or the Miniboard and the XMC microcontroller family in Arduino.

1.1 Required Hardware

Name	Picture
XENSIV™ PAS CO2 Sensor Shield2Go	
or XENSIV™ PAS CO2 Miniboard	
XMC 2Go	
or XMC1100 Boot Kit	
Pin headers (included with the XMC 2Go)	
Micro-USB to USB A cable	

In case of using the miniboard, the following items are also required:

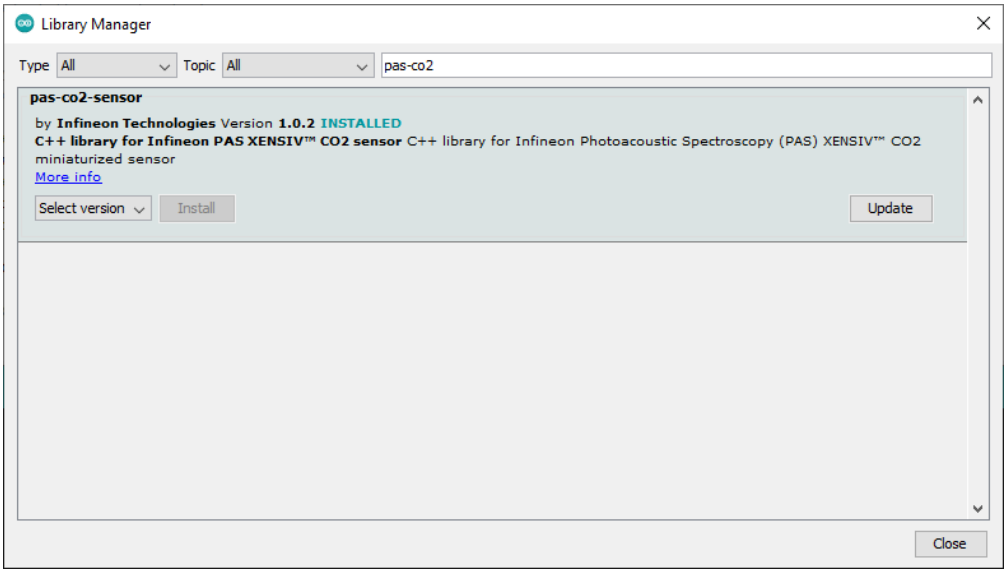
- Jumper cables
- 12V DC power supply

1.2 Required Software

- Segger J-Link
- Arduino IDE
- XMC-for-Arduino
- XENSIV™ PAS CO2 Arduino library

1.3 Software Installation

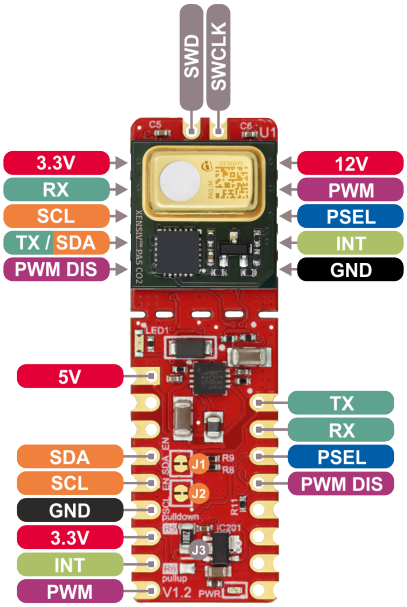
0. **Install Arduino IDE**. If you are new to Arduino, please download the program and install it first.
1. **Install XMC Board**. The official Arduino boards are already available in the Arduino software, but other third party boards as the Infineon XMC MCU based need to be explicitly included. Follow the instructions in the link to add the XMC board family to Arduino. Do not forget to install as well the JLink software.
2. **Install the library**. In the Arduino IDE, go to the menu *Sketch* > *Include library* > *Library Manager*. Type **XENSIV PAS CO2** and install the library.



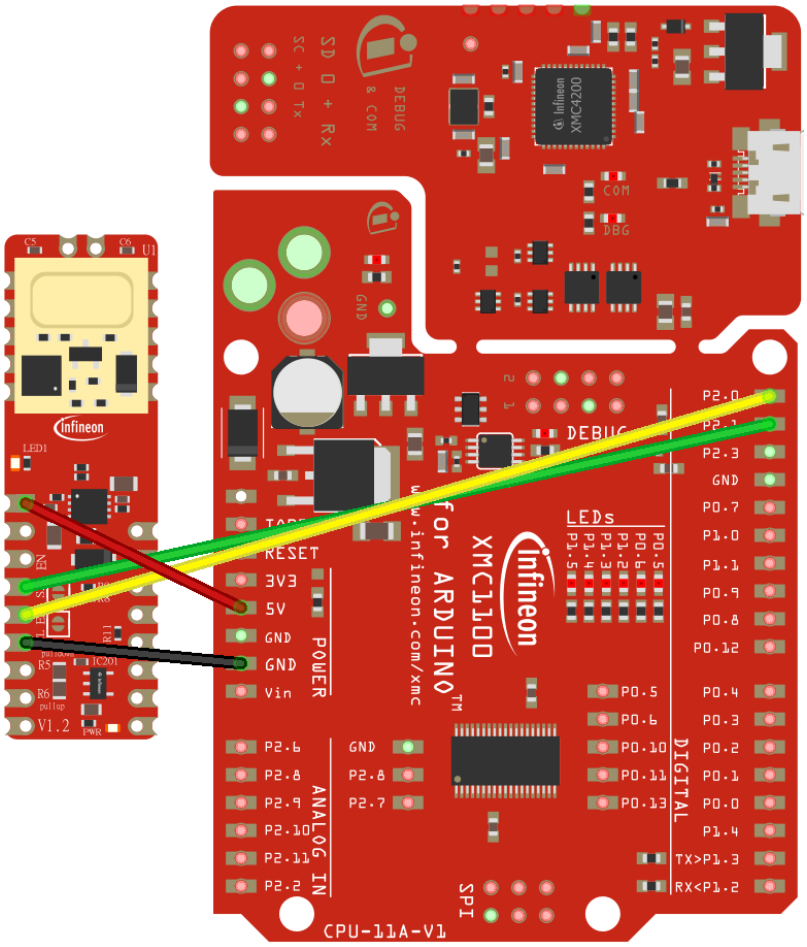
1.4 Hardware Setup

For this example we are going to use the I2C interface.

1.4.1 A. Shield2Go



If you are using the XENSIV™ PAS CO2 Sensor Shield2Go, it is recommended to use a microcontroller which provides a 5V output. In this tutorial, the XMC11000 Boot Kit is used. Connect the shield and the eval kit as in the following wiring diagram:

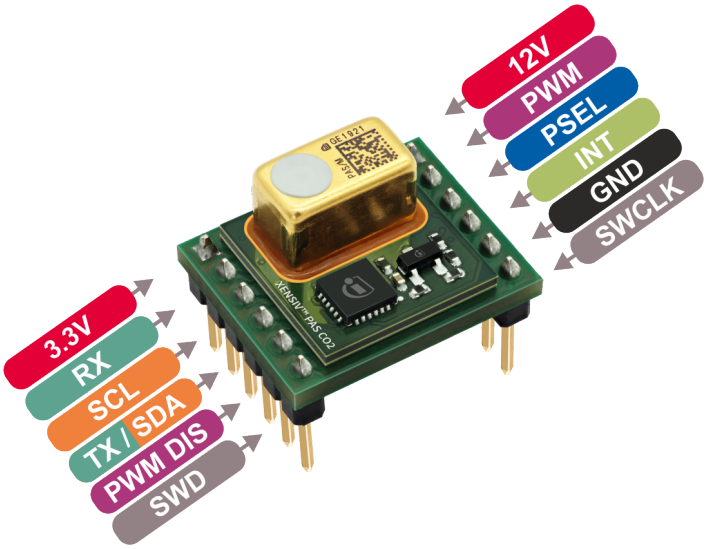


Be sure that the corresponding solder jumper are set for I2C mode, and PSEL is pulled to GND. For this example it is not required, but consider connecting the interrupt signal to the pin 2(P1.4) or 3(P0.0) of the XMC1100 Boot Kit for appliations that require interrupts. Check the Shield2Go Manual for complete details.

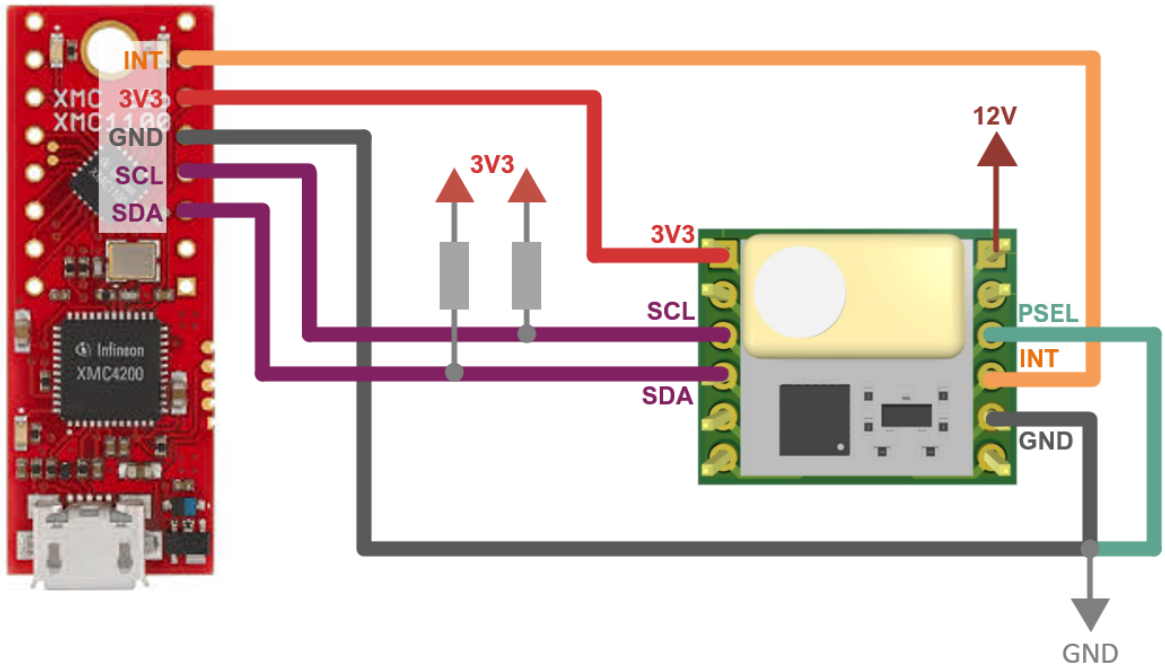
Then, simply connect the eval kit to the computer with the USB cable.

Warning: Alternatively, the XMC 2Go can be used by stackin the sensor shield on top. But the XMC 2Go V1 does not support 5V signal, as required by the XENSIV™ PAS CO2 Sensor Shield2Go. Thus, if XMC 2Go is used, keep in mind that an additional 5V signal needs to be provided to the 5V pin of the XENSIV™ PAS CO2 Sensor Shield2Go.

1.4.2 B. Miniboard



In order to use the I2C interface we need to add a 10 Kohm pull-up resistors to the SDA and SCL lines, and a 12VDC voltage needs to be additionally provided to VDD12V pin. Connect the boards as shown in the following diagram:



You need to provide a 12V DC signal to for the emitter. Then, simply connect it to the computer with the USB cable.

Note : If the pin headers provided are not press-fit you will need to solder them on the corresponding boards. Otherwise, use your preferred way of connecting the hardware.

1.5 Ready To Go!

With everything ready, now we are going to upload and run one of the library examples.

1. **Select the board**

   Once installed the XMC board family, you can select one of the supported board from the menu *Tools* > *Board:*. Choose the **XMC1100 XMC2Go** or **XMC1100 Boot Kit** depending on your hardware setup (*Tools* > *Board* > *XMC Family* > *XMC1100 XMC2Go/XMC1100 Boot Kit*).

2. **Open the example**

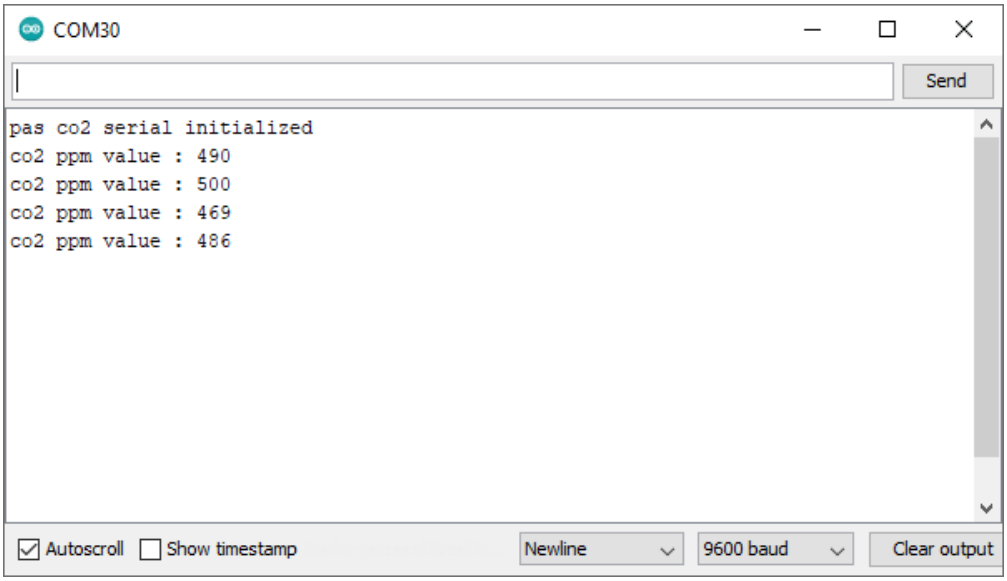
   With the library installed in the Arduino IDE, you can include it from the menu *Sketch* > *Include Library* > *XENSIV PAS CO2*. The header `#include <pas-co2-ino.hpp>` will be added to your sketch. In this case, open and run one of the examples provided in *File* > *Examples* > *XENSIV PAS CO2*.

   Let´s try the continuous mode example: *File* > *Examples* > *XENSIV PAS CO2* > *continuous-mode*.

3. **Build and run the example**

   Select the proper COM port (*Tools* > *Port*), and then verify the example and upload it the target .

   Finally, we can check the monitor output . Do not forget to select the proper baudrate for the serial terminal. You can blow into the sensor to see how the CO2 values change .



1.6 What's next?

This is just the start !

Check out the rest of the available *library examples* and find out more about the library functions in the *API reference* section.

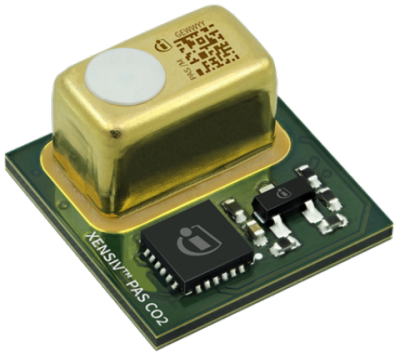
CHAPTER

TWO

HARDWARE PLATFORMS

2.1 Supported Sensor Boards

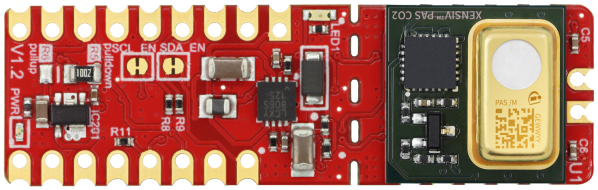
This library can support any break-out board or (PAS) XENSIV™ CO2 Sensor based kit.



- XENSIV™ PAS CO2 Stand-alone module product page
- XENSIV™ PAS CO2 Sensor documentation

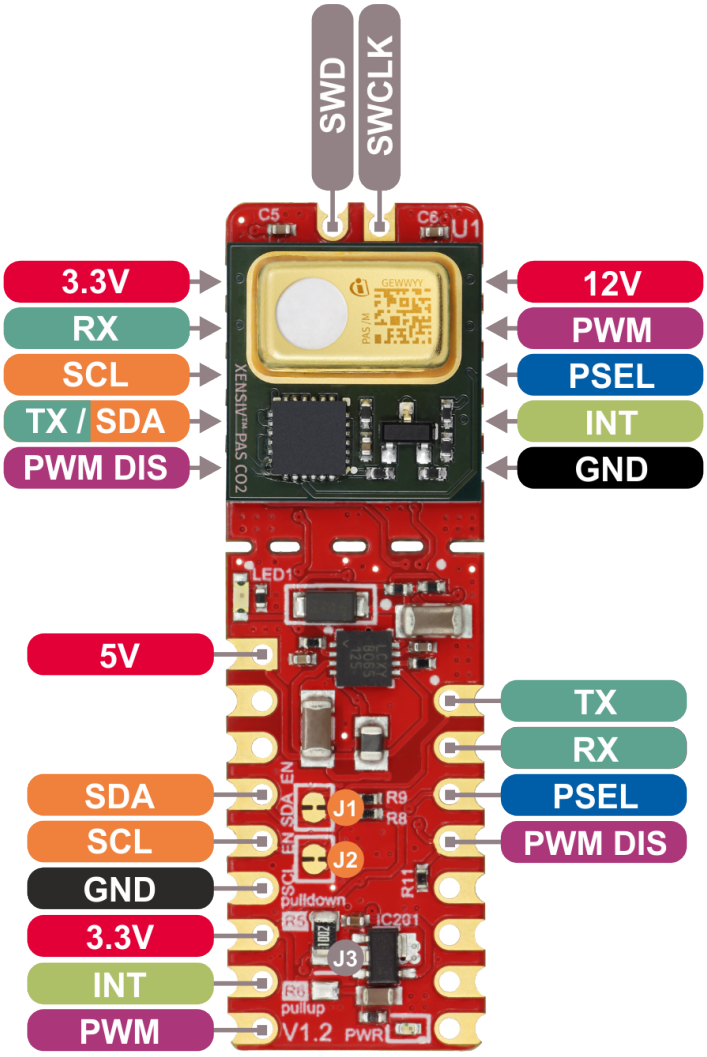
The following kits can be integrated in Arduino compatible MCUs and are supported by this library:

2.1.1 XENSIV™ PAS CO2 Sensor Shield2Go



- XENSIV™ PAS CO2 Shield2Go product page
- Quick Start Guide Shield2Go (for Arduino)

Pinout Diagram

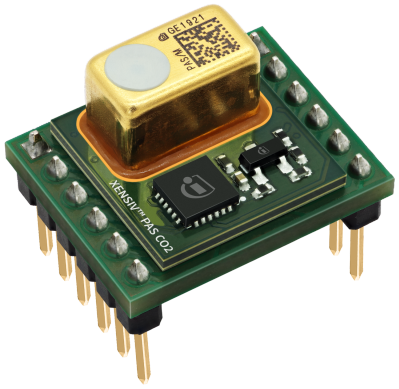


Warning: All signal pins run on 3.3V logic!

Pin Description

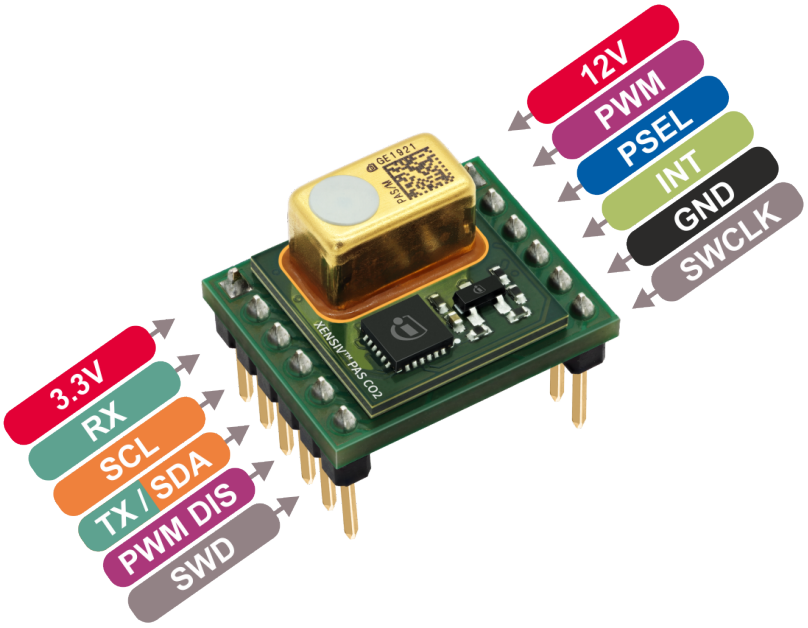
Pin Name	Description
5V	5V supply input.
SDA	I2C SDA (serial data).
SCL	I2C SCL (serial clock).
GND	Supply and signal ground.
3.3V	3.3V supply input - use as logic supply when using breakable part stand-alone, else keep NC.
INT	Interrupt output.
PWM	PWM signal output.
TX	UART transmit side.
RX	UART receive side.
PSEL	Communication interface selection.
PWM DIS	PWM disable input (set high to disable PWM).
12V	12V supply input - use as sensor supply when using breakable part stand-alone, else keep NC.
TX/SDA	UART transmit or I2C SDA (serial data), depending on selected communication interface.
SWD	Serial wire debug data (keep NC).
SWCLK	Serial wire debug clock (keep NC).

2.1.2 XENSIV™ PAS CO2 Miniboard



- XENSIV™ PAS CO2 Miniboard product page
- XENSIV™ PAS CO2 Miniboard documentation

Pinout Diagram



Pin Description

Pin Name	Description
SDA	I2C SDA (serial data).
SCL	I2C SCL (serial clock).
GND	Supply and signal ground.
3.3V	3.3V logic supply input (required).
INT	Interrupt output.
PWM	PWM signal output.
RX	UART receive side.
PSEL	Communication interface selection.
PWM DIS	PWM disable input (set high to disable PWM).
12V	12V sensor supply input (required).
TX/SDA	UART transmit or I2C SDA (serial data), depending on selected communication interface.
SWD	Serial wire debug data (keep NC).
SWCLK	Serial wire debug clock (keep NC).

2.2 Supported MCU Platforms

In principle, the library is supported by any Arduino compatible MCU platform. Its Arduino core needs to implement the Arduino reference language and the Wire built-in Arduino library.

2.2.1 Verified MCU Boards

The library examples have been built and successfully executed on the following hardware platforms:

MCU Platforms
XMC 2Go
XMC1100 Boot Kit
Arduino Uno Rev3

Find out which boards are build checked under continuous integration here.

CHAPTER

THREE

LIBRARY INSTALLATION

The library can be installed in several ways:

- Arduino IDE Library Manager
- Arduino IDE Import .zip library
- Arduino IDE manual installation
- PlatformIO

These installation processes are conveniently described in the official Arduino website.

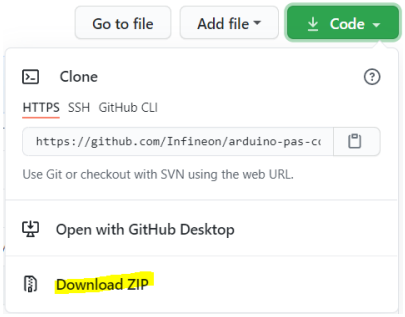
3.1 Library Manager

Library name: `XENSIV PAS CO2`

3.2 Manual Installation

Download the desired .zip library version from the repository releases section.

As a general recommendation, direct downloads from the master branch should be avoided. Even if it should not, it could contain incomplete or faulty code.



3.3 PlatformIO

If you are a PlatformIO user, you have also this library available in the PlatformIO register.

With the project created, now the library and its dependencies can be configured in the ***Platform.ini* Project File**. This file, located in the project root folder, includes one (or several) building environments *[env:__]*.

In the *environment* section, the platform, board, and framework are specified. PlatformIO will take care of downloading and installing all dependencies.

In the following example, we use the XMC 2Go Evaluation Kit (only available for Arduino):

```
[env:xmc1100_xmc2go]
platform = infineonxmc
board = xmc1100_xmc2go
framework = arduino

lib_deps =
    infineon/XENSIV PAS CO2@^3.0.2
```

Find more information in the PlatformIO Registry.

CHAPTER

FOUR

EXAMPLES

The following examples are provided in the library:

alarm-nofitication	Readout of the sensor CO2 concentration based on threshold crossing and synched via hardware interrupt
continuous-mode	Readout of the sensor CO2 concentration value using continuous measurement mode
device-id	Readout of the sensor devices product and revision identifiers
early-notification	Readout of the sensor CO2 concentration based on early notification synched via hardware interrupt
forced-compensation	Set CO2 reference offset using forced compensation
single-shot-mode	Readout of the sensor CO2 concentration value using single shot measurement mode

CHAPTER

FIVE

API REFERENCE

The Arduino library API is implemented via the PASCO2Ino class.

5.1 XENSIV™ PAS CO2 Arduino API

class **PASCO2Ino**

Public Functions

PASCO2Ino(TwoWire *wire = &Wire, uint8_t intPin = *unusedPin*)
XENSIV™ PAS CO2 I2C Arduino Constructor.

Parameters

- **wire** – **[in]** TwoWire interface instance. Default is the Arduino primary Wire instance.
- **intPin** – **[in]** Interrupt pin. Default is UnusedPin

Pre None

PASCO2Ino(HardwareSerial *serial, uint8_t intPin = *unusedPin*)
XENSIV™ PAS CO2 UART Arduino Constructor.

Parameters

- **serial** – **[in]** Serial interface instance
- **intPin** – **[in]** Interrupt pin. Default is UnusedPin

Pre None

~PASCO2Ino()
XENSIV™ PAS CO2 Arduino Destructor.

It disables the sensor and deletes all the dynamically created PAL instances in the constructor

Pre None

Error_t **begin**()
Begins the sensor.

Initializes the serial interface if the initialization is delegated to the PASCO2 class. Sets the I2C freq or UART baudrate to the default values prior the serial interface initialization. Initializes the interrupt pin if used.

Returns XENSIV™ PAS CO2 error code

Pre None

Returns **XENSIV_PASCO2_OK** – if success

Error_t **end**()

Ends the sensor.

Deinitializes the serial interface if the deinitialization is delegated to the PASCO2Ino class. Deinitializes the interrupt pin if used.

Returns XENSIV™ PAS CO2 error code

Pre begin()

Returns **XENSIV_PASCO2_OK** – always

Error_t **startMeasure**(int16_t periodInSec = 0, int16_t alarmTh = 0, void (*cback)(void*) = nullptr, bool earlyNotification = false)

Triggers the internal measuring of the sensor.

The function start the measurement controlling the different sensor modes and features depending on the configured arguments.

Single shot

If the function is called with no arguments, the sensor will be triggered to perform a single shot measurement. The user needs to poll with getCO2() until the CO2 value is available and has been read out from the sensor. The CO2 concentration value read will be zero as long as no value is available or if any error occurred in the readout attempt. Polling example:

```
PASCO2Ino cotwo(serial_intf);
int16_t   co2ppm;

serial_intf.begin();

cotwo.begin();

cotwo.startMeasure();

do{ cotwo.getCO2(co2ppm); } while (co2ppm == 0);
```

Continuous measurement

Continuous measurements (periodInSec) will configure the sensor to perform a measurement every desired period. Between 5 and 4095 seconds. Without further arguments, the user has to poll with getCO2() until the value is available. Any super loop or thread routine, can just consists on reading the CO2 (getCO2()). For example, measure every 5 minutes:

```
PASCO2Ino cotwo(serial_intf);
int16_t   co2ppm;

serial_intf.begin();

cotwo.begin();

cotwo.startMeasure(300);
```

(continues on next page)

(continued from previous page)

```
while(1)
{
    delay(300000); // Measure will be ready every 5 min

    do{ cotwo.getCO2(co2ppm); } while (co2ppm == 0);
    // ... do something with the co2 value ...
}
```

Synching readouts with the hardware interrupt

In order not to saturate the sensor with constant serial requests, especially in continuous mode, it is recommended to synch the readout with a timer. Or even better using the hardware GPIO hardware interrupt. If the interrupt pin has been provided, passing a callback function will enable the interrupt mode. The type of interrupt is decided depending on the value of the rest of the arguments and operations modes. Some example:

```
volatile bool intFlag = false;
void cback(void *)
{
    intFlag = true;
}

PASCO2Ino cotwo(serial_intf, interrupt);
int16_t   co2ppm;

serial_intf.begin();

cotwo.begin();

cotwo.startMeasure(300,0,cback);

while(1)
{
    while(!intFlag) { // block or yield() };
    cotwo.getCO2(co2ppm);
    // ... do something with the co2 value ...
    intFlag = false;
}
```

Alarm mode

If the alarm threshold argument is non-zero, the alarm mode is activated, and the sensor internal flag will be enabled if the concentration of CO2 goes above the specified value. This option is better combined with the interupt mode. Thus, if the interrupt mode is available and a callback function is passed, the interrupt will occurr only when the co2 concentration goes above the threshold. This makes mostly sense for continuous measurement configuration. But it can be used as well for a single shot configuration

Early notification

The early notification mode can be used for battery power solutions. The interrupt signal can trigger the enablement of the 12V emitter power supply just before the measurement is performed, and switch it off as the interrupt signal is disabled. Therefore, the power supply 12V only needs to be on during the CO2

sensing.

When this flag is set, the alarm interrupt functionality is not available. Both options cannot be combined.

Parameters

- **periodInSec** – **[in]** Enables continuous measurement with the specified period. The default value is 0, meaning single shot operation. The valid period range goes between 5 and 4095 seconds
- **alarmTh** – **[in]** Enables upper alarm threshold mode for the specified ppm value The default value is 0, meaning no alarm mode. For any non-zero value, the sensor will internally set the alarm flag. If an interrupt callback function is provided, then the interrupt will occurr only when the defined threshold has been tresspassed
- **cback** – **[in]** Pointer to the callback function to be called upon interrupt
- **earlyNotification** – **[in]** Enables early notifification interrupt. Disabled (false) by default

Returns XENSIV™ PAS CO2 error code

Pre begin()

Returns **XENSIV_PASCO2_OK** – if success

Error_t **stopMeasure**()

Stops the internal measuring of the sensor.

Sets operation mode to idle

Returns XENSIV™ PAS CO2 error code

Pre begin()

Returns **XENSIV_PASCO2_OK** – if success

Error_t **getCO2**(int16_t &CO2PPM)

Gets the CO2 concentration measured.

The value read is zero when no measurement is yet available or an error has ocurrred.

Parameters **co2ppm** – **[out]** CO2 concentration read (in ppm)

Returns XENSIV™ PAS CO2 error code

Pre startMeasure()

Returns **XENSIV_PASCO2_OK** – if success

Error_t **getDiagnosis**(*Diag_t* &diagnosis)

Gets diagnosis information.

The sensor status registers includes the following flags:

- Sensor ready
- PWM pin enabled
- Temperature out of range error
- IR emitter voltage out of range error
- Communication error which will be stored in the Diag_t struct varible passed by argument. After reading the flags, these are cleared in the device writing in the corresponding clear flag bitfields.

Parameters **diagnosis** – **[out]** Struct to store the diagnosis flags values

Returns XENSIV™ PAS CO2 error code

Pre None

Returns **XENSIV_PASCO2_OK** – if success

Error_t **setABOC**(*ABOC_t* aboc, int16_t abocRef)
Configures the sensor automatic baseline compensation.

Parameters

- **aboc** – **[in]** Automatic baseline compenstation mode
- **abocRef** – **[in]** Automatic baseline compensation reference

Returns XENSIV™ PAS CO2 error code

Pre begin()

Returns **XENSIV_PASCO2_OK** – if success

Error_t **setPressRef**(uint16_t pressRef)
Sets the sensor pressure reference.

Parameters **pressRef** – **[in]** Pressure reference value. Min value is 750, and max 1150.

Returns XENSIV™ PAS CO2 error code

Pre begin()

Returns **XENSIV_PASCO2_OK** – if success

Error_t **performForcedCompensation**(uint16_t co2Ref)
Performs force compensation.

Calculates the offset compensation when the sensor is exposed to a CO2 reference value.

> **Warning:** The device is left in idle mode after the compensation value is stored in non-volatile memory.

Parameters **co2Ref** – **[in]** Automatic baseline compenstation mode

Returns XENSIV™ PAS CO2 error code

Pre begin()

Returns **XENSIV_PASCO2_OK** – if success

Error_t **clearForcedCompensation**()
Resets the forced calibration correction factor.

Returns XENSIV™ PAS CO2 error code

Pre begin()

Returns **XENSIV_PASCO2_OK** – if success

Error_t **reset**()
Resets the sensor via serial command.

Returns XENSIV™ PAS CO2 error code

Pre begin()

Returns **XENSIV_PASCO2_OK** – if success

Error_t **getDeviceID**(uint8_t &prodID, uint8_t &revID)
Gets device product identifier.

Parameters

- **prodID** – **[out]** Product identifier
- **revID** – **[out]** Version identifier

Returns XENSIV™ PAS CO2 error code

Pre begin()

Returns **XENSIV_PASCO2_OK** – if success

Error_t **getRegister**(uint8_t regAddr, uint8_t *data, uint8_t len)
Reads from the sensor device into the given data buffer.

Parameters

- **regAddr** – **[in]** Start register address
- **data** – **[out]** Pointer to the data buffer to store the register values of the sensor
- **len** – **[in]** Number of bytes of data to be read

Returns XENSIV™ PAS CO2 error code

Pre begin()

Returns **XENSIV_PASCO2_OK** – if success

Error_t **setRegister**(uint8_t regAddr, const uint8_t *data, uint8_t len)
Writes the given data buffer into the sensor device.

Parameters

- **regAddr** – **[in]** Start register address
- **data** – **[in]** Pointer to the data buffer to be written in the sensor
- **len** – **[in]** Number of bytes of data to be written

Returns XENSIV™ PAS CO2 error code

Pre begin()

Returns **XENSIV_PASCO2_OK** – if success

Public Static Attributes

static constexpr uint8_t **unusedPin** = 0xFFU
Unused pin

5.1.1 Types

Return Error Codes

typedef int32_t **Error_t**

XENSIV_PASCO2_OK
Result code indicating a successful operation

XENSIV_PASCO2_ERR_COMM
Result code indicating a communication error

XENSIV_PASCO2_ERR_WRITE_TOO_LARGE
Result code indicating that an unexpectedly large I2C write was requested which is not supported

XENSIV_PASCO2_ERR_NOT_READY
Result code indicating that the sensor is not yet ready after reset

XENSIV_PASCO2_ICCERR
Result code indicating whether a non-valid command has been received by the serial communication interface

XENSIV_PASCO2_ORVS
Result code indicating whether a condition where VDD12V has been outside the specified valid range has been detected

XENSIV_PASCO2_ORTMP
Result code indicating whether a condition where the temperature has been outside the specified valid range has been detected

XENSIV_PASCO2_READ_NRDY
Result code indicating that a new CO2 value is not yet ready

Dignosis

typedef *xensiv_pasco2_status_t* **Diag_t**

union **xensiv_pasco2_status_t**
#include <xensiv_pasco2.h> Structure of the sensor's status register (SENS_STS)

Public Members

uint32_t **__pad0__**

uint32_t **iccerr**
Communication error notification bit. Indicates whether an invalid command has been received by the serial communication interface

uint32_t **orvs**
Out-of-range VDD12V error bit

uint32_t **ortmp**
: Out-of-range temperature error bit

uint32_t **pwm_dis_st**
: PWM_DIS pin status

uint32_t **sen_rdy**
: Sensor ready bit

struct *xensiv_pasco2_status_t*::[anonymous] **b**
: Structure used for bit access

uint8_t **u**
: Type used for byte access

Baseline Offset Compensation

typedef *xensiv_pasco2_boc_cfg_t* **ABOC_t**

enum **xensiv_pasco2_boc_cfg_t**
: Enum defining the different device baseline offset compensation (BOC) modes

Values:

enumerator **XENSIV_PASCO2_BOC_CFG_DISABLE**
: No offset compensation occurs

enumerator **XENSIV_PASCO2_BOC_CFG_AUTOMATIC**
: The offset is periodically updated at each BOC computation

enumerator **XENSIV_PASCO2_BOC_CFG_FORCED**
: Forced compensation

5.2 XENSIV™ PAS CO2 C Reference API

The Arduino library is wrapping the platform abstracted C library from this project. Find out the complete C core library documentation here.

CHAPTER

SIX

LICENSE

Find the license for this library here.

INDEX

A

ABOC_t (*C++ type*), 26

D

Diag_t (*C++ type*), 25

E

Error_t (*C++ type*), 25

P

PASCO2Ino (*C++ class*), 19
PASCO2Ino::~PASCO2Ino (*C++ function*), 19
PASCO2Ino::begin (*C++ function*), 19
PASCO2Ino::clearForcedCompensation (*C++ function*), 23
PASCO2Ino::end (*C++ function*), 20
PASCO2Ino::getCO2 (*C++ function*), 22
PASCO2Ino::getDeviceID (*C++ function*), 24
PASCO2Ino::getDiagnosis (*C++ function*), 22
PASCO2Ino::getRegister (*C++ function*), 24
PASCO2Ino::PASCO2Ino (*C++ function*), 19
PASCO2Ino::performForcedCompensation (*C++ function*), 23
PASCO2Ino::reset (*C++ function*), 23
PASCO2Ino::setABOC (*C++ function*), 23
PASCO2Ino::setPressRef (*C++ function*), 23
PASCO2Ino::setRegister (*C++ function*), 24
PASCO2Ino::startMeasure (*C++ function*), 20
PASCO2Ino::stopMeasure (*C++ function*), 22
PASCO2Ino::unusedPin (*C++ member*), 24

X

xensiv_pasco2_boc_cfg_t (*C++ enum*), 26
xensiv_pasco2_boc_cfg_t::XENSIV_PASCO2_BOC_CFG_AUTOMATIC (*C++ enumerator*), 26
xensiv_pasco2_boc_cfg_t::XENSIV_PASCO2_BOC_CFG_DISABLE (*C++ enumerator*), 26
xensiv_pasco2_boc_cfg_t::XENSIV_PASCO2_BOC_CFG_FORCED (*C++ enumerator*), 26
XENSIV_PASCO2_ERR_COMM (*C macro*), 25
XENSIV_PASCO2_ERR_NOT_READY (*C macro*), 25
XENSIV_PASCO2_ERR_WRITE_TOO_LARGE (*C macro*), 25
XENSIV_PASCO2_ICCERR (*C macro*), 25
XENSIV_PASCO2_OK (*C macro*), 25
XENSIV_PASCO2_ORTMP (*C macro*), 25
XENSIV_PASCO2_ORVS (*C macro*), 25
XENSIV_PASCO2_READ_NRDY (*C macro*), 25
xensiv_pasco2_status_t (*C++ union*), 25
xensiv_pasco2_status_t::__pad0__ (*C++ member*), 25
xensiv_pasco2_status_t::b (*C++ member*), 26
xensiv_pasco2_status_t::iccerr (*C++ member*), 25
xensiv_pasco2_status_t::ortmp (*C++ member*), 25
xensiv_pasco2_status_t::orvs (*C++ member*), 25
xensiv_pasco2_status_t::pwm_dis_st (*C++ member*), 26
xensiv_pasco2_status_t::sen_rdy (*C++ member*), 26
xensiv_pasco2_status_t::u (*C++ member*), 26