SENSIRION
THE SENSOR COMPANY

SGP30 Driver Integration (for Dedicated I²C Hardware)

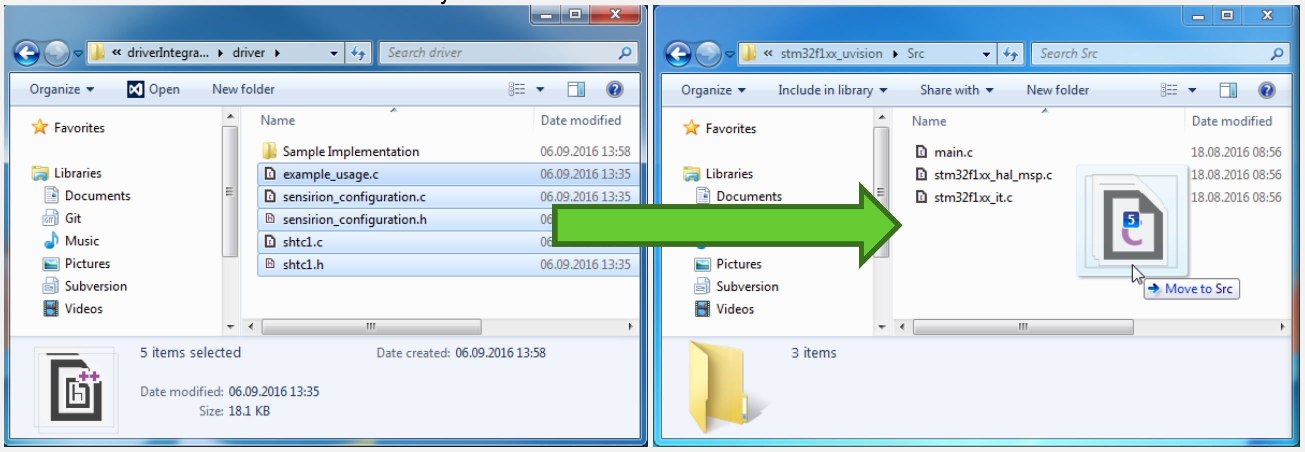
A Step-by-Step Guide

Preface

The easiest way to integrate the SGP30 sensor into a device is Sensirion's SGP30 driver. This document explains how to implement the hardware abstraction layer of the SGP driver and describes the provided API.

COPY FILES TO YOUR PROJECT — STEP 1 2 3

1.1. Copy all SGP driver files (.c and .h) into your software project folder.
1.2. Make sure all files are added to your IDE



```
Name                           Date modified
Sample Implementation          06.09.2016 13:58
example_usage.c                06.09.2016 13:35
sensirion_configuration.c      06.09.2016 13:35
sensirion_configuration.h      06
shtc1.c                        06
shtc1.h                        06.09.2016 13:35

5 items selected     Date created: 06.09.2016 13:58
Date modified: 06.09.2016 13:35
Size: 18.1 KB
```

```
Name                           Date modified
main.c                         18.08.2016 08:56
stm32f1xx_hal_msp.c            18.08.2016 08:56
stm32f1xx_it.c                 18.08.2016 08:56

3 items
```

IMPLEMENT sensirion_configuration.c — 1 STEP 2 3

To use your I²C hardware the file **sensirion_configuration.c** (or *sensirion_configuration.cpp* for c++ projects) needs to be completed. All parts marked with "`// IMPLEMENT`" have to be replaced with code performing the necessary setup.

2.1. Implement the I²C initialization for your specific hardware.

```
void sensirion_i2c_init()
{
    // IMPLEMENT
}
```

2.2. Implement `sensirion_i2c_read()`, which executes a read command on the I²C bus, reading the given number of bytes. The specified `address` is the address of the SGP sensor. Write the number of read bytes (`count`) into the given `data` buffer.

```
int8_t sensirion_i2c_read(uint8_t address, uint8_t* data, uint16_t count)
{
    // IMPLEMENT
    return 0;
}
```

Return: 0 if read command is executed successfully, else an error code.

2.3. Implement `sensirion_i2c_write()`, which executes a write command on the I²C bus. The specified `address` is the address of the SGP sensor. Write the given number of bytes (`count`) from the buffer `data` to the I²C bus.

```
int8_t sensirion_i2c_write(uint8_t address, const uint8_t *data, uint16_t count)
{
    // IMPLEMENT
    return 0;
}
```

Note: Some implementations of I²C write/read expect an 8 bit sensor address (instead of 7 bit). In this case use `(address<<1)` instead of `address` in your implementation.

Return: 0 if the write command is executed successfully, else an error code.

2.4. Implement `sensirion_sleep_usec()`, which delays the execution for the given time in microseconds.

```
void sensirion_sleep_usec(uint32_t useconds) {
    // IMPLEMENT
}
```

MEASURE IAQ (tVOC / CO2eq) AND SIGNAL VALUES

1 2 STEP 3

The SGP driver provides functions to probe the sensor, to get the serial ID, and to measure/read tVOC and CO2-eq.

3.1. Call `sgp_probe()` to initialize the I²C bus and test if the sensor is available.

```
int16_t sgp_probe(void);
```

Return: 0 if the sensor is detected, else an error code.

3.2. Call `sgp_get_serial_id ()` to readout the serial id of the SGP sensor.

```
int16_t sgp_get_serial_id (u64 *serial_id);
```

Return: 0 if the command is successful, else an error code.

3.3. Call `sgp_get_feature_set_version()` to readout the feature set version and product type of the SGP sensor. If product_type is 0 it is a SGP30 gas sensor, if it is 1 it is an SGPC3 gas sensor.

```
int16_t sgp_get_feature_set_version (u16 *feature_set_version, u8 *product_type);
```

Return: 0 if the command is successful, else an error code.

3.4. Call `sgp_measure_iaq_blocking_read()` to start a tVOC and CO2-eq measurement and to readout the values.

```
int16_t sgp_measure_iaq_blocking_read(uint16_t *tvoc_ppb, uint16_t *co2_eq_ppm);
```

Note: This function blocks the processor while the measurement is in progress.
Return: 0 if the command is successful, else an error code.

3.5. For non-blocking measurement and readout of tVOC and CO2-eq use the two functions `sgp_measure_iaq()` and `sgp_read_iaq()`.

```
int16_t sgp_measure_iaq(void);
```

Return: 0 if the command is successful, else an error code.

Be sure to wait until the SGP sensor finishes the measurement. Read the measurement result using `sgp_read_iaq()`.

```
int16_t sgp_read_iaq(uint16_t *tvoc_ppb, uint16_t *co2_eq_ppm);
```

Note: If the measurement is still in progress, this function returns an error code.
Return: 0 if the command is successful, else an error code.

3.6. For best performance and faster startup times, the current baseline needs to be persistently stored on the device before shutting it down and set again accordingly after boot up.

Use `sgp_get_iaq_baseline()` to get the baseline.

```
int16_t sgp_get_iaq_baseline (uint32_t *baseline);
```

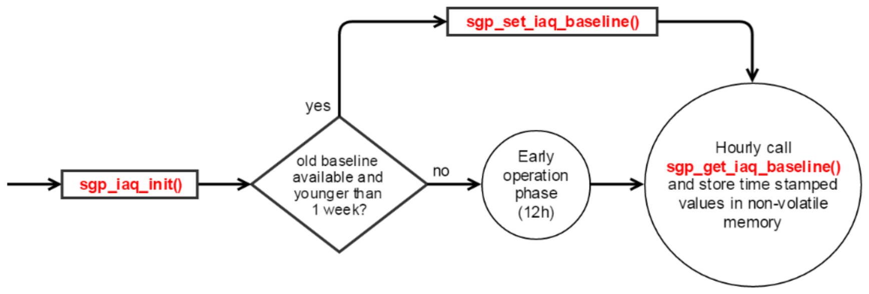
Return: 0 if the command is successful, else an error code.
Note: If the call is not successful, the `baseline` value must be discarded. Approximately in the first 15 seconds of operation after `sgp_probe` or `sgp_iaq_init` the call will fail unless a previous baseline was restored.

3.7. Use `sgp_set_iaq_baseline()` to set the baseline.

```
int16_t sgp_set_iaq_baseline (uint32_t baseline);
```

Return: 0 if the command is successful, else an error code.
Note: The baseline value must be *exactly* as returned by `sgp_get_iaq_baseline()` and should only be set if it's less than one week old.

3.8. SGP baseline states


sgp_set_iaq_baseline()
yes
sgp_iaq_init()
old baseline available and younger than 1 week?
no
Early operation phase (12h)
Hourly call sgp_get_iaq_baseline() and store time stamped values in non-volatile memory

Call `sgp_iaq_init()` to reset all SGP baselines. The initialization takes up to around 15 seconds, during which `sgp_measure_iaq()` output will not change.

If no stored baseline is available after initializing the baseline algorithm, the sensor has to run for 12 hours until the baseline can be stored. This will ensure an optimal behavior for subsequent startups. Reading out the baseline prior should be avoided unless a valid baseline is restored first. Once the baseline is properly initialized or restored, the current baseline value should be stored approximately once per hour. While the sensor is off, baseline values are valid for a maximum of seven days.

3.9. Call `sgp_iaq_init()` to initialize or re-initialize the indoor air quality algorithm.

```
int16_t sgp_iaq_init(void);
```

Return: 0 if the command is successful, else an error code.
Note: `sgp_iaq_init()` is already called as part of `sgp_probe()`.

3.10. Call `sgp_measure_tvoc_blocking_read()` to start a tVOC measurement and to readout the value in ppb.

```
int16_t sgp_measure_tvoc_blocking_read(uint16_t *tvoc_ppb);
```

Note: This function blocks the processor while the measurement is in progress.

Return: 0 if the command is successful, else an error code.

3.11. For non-blocking measurement and readout of tVOC use the two functions `sgp_measure_tvoc()` and `sgp_read_tvoc()`

```
int16_t sgp_measure_tvoc(void);
```

Return: 0 if the command is successful, else an error code.

Be sure to wait until the SGP sensor finishes the measurement. Read the measurement result using `sgp_read_tvoc()`.

```
int16_t sgp_read_tvoc(uint16_t *tvoc_ppb);
```

Note: If the measurement is still in progress, this function returns an error code.

Return: 0 if the command is successful, else an error code.

3.12. Call `sgp_measure_co2_eq_blocking_read()` to start a CO2-eq measurement and to readout the value in ppm.

```
int16_t sgp_measure_co2_eq_blocking_read(uint16_t *co2_eq_ppm);
```

Note: This function blocks the processor while the measurement is in progress.
Return: 0 if the command is successful, else an error code.

3.13. For non-blocking measurement and readout of CO2-eq use the two functions `sgp_measure_co2_eq()` and `sgp_read_co2_eq()`.

```
int16_t sgp_measure_co2_eq(void);
```

Return: 0 if the command is successful, else an error code.

Be sure to wait until the SGP sensor finishes the measurement. Read the measurement result using `sgp_read_co2_eq()`.

```
int16_t sgp_read_co2_eq(uint16_t *co2_eq_ppm);
```

Note: If the measurement is still in progress, this function returns an error code.
Return: 0 if the command is successful, else an error code.

3.14. Call `sgp_measure_signals_blocking_read()` to start signal measurements and to readout the values.

```
int16_t sgp_measure_signals_blocking_read(uint16_t *ethanol_signal,
                                          uint16_t *h2_signal);
```

Return: 0 if the command is successful, else an error code.

Note: This function blocks the processor while the measurement is in progress.

3.15. For non-blocking measurement and readout of signals values use the two functions `sgp_measure_signals()` and `sgp_read_signals()`.

```
int16_t sgp_measure_signals(void);
```

Return: 0 if the command is successful, else an error code.

Be sure to wait until the SGP sensor finishes the measurement. Read the measurement results using `sgp_read_signals()`.

```
int16_t sgp_read_signals(uint16_t * ethanol_signal,
                         uint16_t * h2_signal);
```

Note: If the measurement is still in progress, this function returns an error code.
Return: 0 if the command is successful, else an error code.

3.16. Call `sgp_set_absolute_humidity()` to a value greater than 0 and smaller than 256000 mg/m³ to enable the humidity compensation feature, or write 0 to disable it.
The absolute humidity in g/m³ can be retrieved by measuring the relative humidity and temperature using a Sensirion SHT sensor and converting the value to absolute humidity with the formula

$$AH = 216.7 \cdot \frac{\frac{RH}{100.0} \cdot 6.112 \cdot \exp\frac{17.62 \cdot t}{243.12+t}}{273.15+t}$$

With *AH* in g/m³, *RH* in 0-100%, and *t* in °C

Note: the value in g/m³ has to be multiplied by 1000 to convert to mg/m³ and any remaining decimal places have to be rounded and removed since the interface does not support floating point numbers.

```
int16_t sgp_set_absolute_humidity(uint32_t absolute_humidity);
```

Return: 0 if the command is successful, else an error code.

Note: The humidity compensation is disabled by setting the value to 0.
Example: To set the absolute humidity to 13.000 g/m³:

```
// Set absolute humidity to 13.000 g/m^3
uint32_t ah = 13000;
sgp_set_absolute_humidity(ah);
```

3.17. Call `sgp_measure_test()` to run the on-chip self-test. This command can be used during production to ensure the SGP30 is not damaged. A success is indicted by a return code of 0, in which case the value of `test_result` is *0xd400*.

```
// Run the on-chip self-test
uint16_t test_result;
int16_t ret = sgp_measure_test(&test_result);
if (ret != STATUS_OK) {
    // The sensor is likely damaged
}
```

Note: **`sgp_measure_test()` must not be executed after `sgp_iaq_init()`**. If this is needed, the baseline must be retrieved prior to running `sgp_measure_test`. After `sgp_measure_test`, `sgp_iaq_init` followed by setting the baseline again is needed to resume IAQ operations.

3.18. Call `sgp_get_driver_version()` to retrieve the driver version.

```
const char *sgp_get_driver_version(void);
```

Return: The driver version string is returned in the form "major.minor.patchset" e.g. "2.2.1"

3.19. Call `sgp_get_tvoc_inceptive_baseline()` to retrieve the on-chip inceptive baseline and call `sgp_set_tvoc_baseline()` to set it.
The inceptive baseline may only be used on the very first startup of the sensor. It ensures that measured concentrations are consistent with the air quality even before the first clean air event.

The command chain to use the inceptive baseline is thus:

```
s16 ret;
u16 tvoc_inceptive_baseline;
ret = sgp_get_tvoc_inceptive_baseline(&tvoc_inceptive_baseline);
if (ret == STATUS_OK) {
    ret = sgp_set_tvoc_baseline(tvoc_inceptive_baseline);
    if (ret != STATUS_OK) { /* error */ }
}
```

Note: The inceptive baseline may only be used on the very first startup of the sensor.
The inceptive baseline is available on SGP30 with feature set >= 1.1 (0x0021) and SGPC3 with feature set >= 0.5 (0x1005). An error is reported when trying to read the inceptive baseline from a sensor that does not support it.

REVISION HISTORY

Date	Version	Page(s)	Changes
October 2016	1.0.0	all	Initial release
January 2017	1.0.1	all	Add CO2-eq output to the driver
January 2017	1.0.2	all	Fixing layout
January 2017	1.1.0	all	Add IAQ functions
January 2017	1.1.1	3	Document how long a baseline value is valid
January 2017	1.2.0	3-5	Document baseline documentation
March 2017	1.4.0	3	Update baseline persistence documentation
April 2017	1.5.0	1, 3	SGP30
Mai 2017	2.0.0	all	Change interfaces from resistance to ethanol and h2 signals
Mai 2017	2.0.1	all	Document signal scaling
August 2017	2.1.0	6	Add humidity compensation, measure_test
August 2017	2.1.1	6	Document driver version
September 2017	2.2.0	2	Document sgp_get_serial_id interface
September 2017	2.2.1	4, 5	Remove outdated notes
April 2018	2.3.0	4 ,5	Signals are not scaled by 512 anymore
July 2018	2.4.0	6	Factory Baseline
February 2019	2.4.1	6	Rename factory into inceptive baseline

Headquarters and Subsidiaries

Sensirion AG
Laubisruetistr. 50
CH-8712 Staefa ZH
Switzerland

Phone: +41 44 306 40 00
Fax: +41 44 306 40 30
info@sensirion.com
www.sensirion.com

Sensirion Inc., USA
Phone: +1 805 409 4900
info_us@sensirion.com
www.sensirion.com

Sensirion Japan Co. Ltd.
Phone: +81 3 3444 4940
info@sensirion.co.jp
www.sensirion.co.jp

Sensirion Korea Co. Ltd.
Phone: +82 31 345 0031 3
info@sensirion.co.kr
www.sensirion.co.kr

Sensirion China Co. Ltd.
Phone: +86 755 8252 1501
info@sensirion.com.cn
www.sensirion.com.cn

Sensirion AG (Germany)
Phone: +41 44 927 11 66
info@sensirion.com
www.sensirion.com

To find your local representative, please visit www.sensirion.com/contact