SGPC3 Driver Integration (for Software I²C)

A Step-By-Step Guide

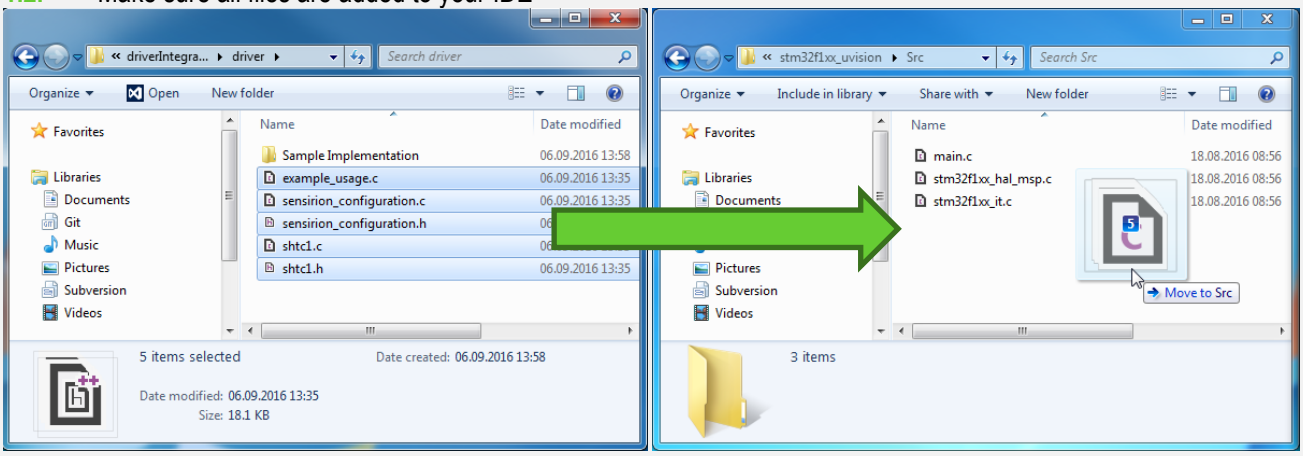
Preface

The easiest way to integrate the SGPC3 sensor into a device is Sensirion's SGPC3 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



IMPLEMENT sensirion_sw_i2c_implementation.c — 1 STEP 2 3

To use software I²C the file **sensirion_sw_i2c_implementation.c** and **sensirion_arch_config.h** need to be completed. All parts marked with "`// IMPLEMENT`" have to be replaced with the according setup.

2.1. Initialize the components needed to set SDA and SCL pins.

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

2.2. Implement `sensirion_SDA_in()` and `sensirion_SCL_in()` to configure the SDA and SCL pins as input.

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

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

2.3. Implement `sensirion_SDA_out()` and `sensirion_SCL_out()` to configure the pins as output.

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

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

2.4. Implement `sensirion_SDA_read()` and `sensirion_SCL_read()` to read the values from the pins.

```
uint8_t sensirion_SDA_read()
{
    // IMPLEMENT
    return 1;
}

uint8_t sensirion_SCL_read()
{
    // IMPLEMENT
    return 1;
}
```

Return: 0 if the pin is low and 1 otherwise.

2.5. Implement `sensirion_sleep_usec()` to delay the execution for the given time in microseconds.

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

Note: the required precision depends on the i2c bus frequency.

MEASURE IAQ (tVOC) 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.

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 measurement and to readout the values.

```
int16_t sgp_measure_iaq_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.

3.5. For non-blocking measurement and readout of tVOC 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);
```

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 (uint16_t *baseline);
```

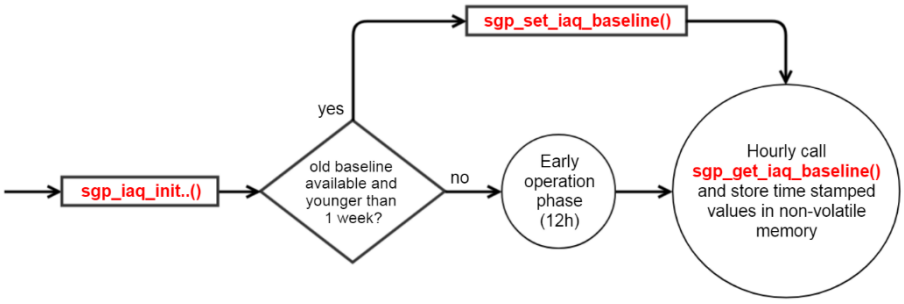
Return: 0 if the command is successful, else an error code.
Note: If the call is not successful, the `baseline` value must be discarded. Approximately in the first 60min of operation after `sgp_probe` or `sgp_iaq_init_continuous` the call will fail unless a previous baseline was restored.

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

```
int16_t sgp_set_iaq_baseline (uint16_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



Call `sgp_iaq_init_continuous()` to reset all SGP baselines. After the accelerated warm-up phase (see 3.9. for durations), the initialization takes 20 seconds, during which the 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 the initialization function `sgp_iaq_init_continuous()` to initialize or re-initialize the indoor air quality algorithm.

Sensor down-time	Recommended initialization	Accelerated warm-up

1 min – 30 min	`sgp_iaq_init_continuous();`
	`sgp_set_iaq_baseline();`
	`sgp_measure_tvoc_blocking_read();`
		-

30 min – 6 h	`sgp_iaq_init_continuous();`
	`sgp_set_iaq_baseline();`
	`sleep(16);`
	`sgp_measure_tvoc_blocking_read();`
		16 s

6 h – 1 week	`sgp_iaq_init_continuous();`
	`sgp_set_iaq_baseline();`
	`sleep(184);`
	`sgp_measure_tvoc_blocking_read()`
		184 s

more than 1 week
/ initial switch-on
	`sgp_iaq_init_continuous();`
	`sleep(184);`
	`sgp_measure_tvoc_blocking_read()`
		184 s

The above initialization logic ensures fast switch-on behavior for typical IAQ applications. It can be modified and optimized according to the specific limitations and requirements of individual devices.

```
int16_t sgp_iaq_init_continuous(void);
```

Return: 0 if the command is successful, else an error code.
Note: `sgp_iaq_init_continuous()` is already called as part of `sgp_probe()`.
Note: The accelerated warm-up phase is interrupted by the first IAQ measurement. The caller must thus sleep for the desired amount of accelerated warm-up time before measuring IAQ with `sgp_measure_tvoc()` or `sgp_measure_tvoc_blocking_read()`.

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_signals_blocking_read()` to start signal measurements and to readout the values.

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

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

```
uint16_t ret;
uint16_t ethanol_signal;

// run signals measurement
ret = sgp_measure_signals_blocking_read(&ethanol_signal);
```

3.13. 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.

```
uint16_t ret;
uint16_t ethanol_signal;

// run signals measurement
ret = sgp_measure_signals();
usleep(200000);
sgp_read_signals(&ethanol_signal);
```

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);
```

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. From feature set 6 on, the SGPC3 features an ultra-low power mode. Run `sgp_set_power_mode()` to change the power mode. Valid power modes are 0 (ultra-low power mode with 30s measurement interval) and 1 (low power mode with 2s measurement interval). 1 (low power mode) is the sensor's startup or reset default.

```
int16_t sgp_set_power_mode(uint16_t power_mode);
```

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

Note: IAQ must be reinitialized after changing the power mode. Also, the IAQ baseline is tied to the power mode. Only set a baseline that was acquired in the same power mode.

Note: In ultra-low power mode IAQ must be initialized with `sgp_iaq_init_continuous()`.

```
uint16_t ret;
uint16_t tvoc_ppb;

// Set the Ultra-Low Power mode
ret = sgp_set_power_mode(0);

// reinitialize IAQ
```

```
ret = sgp_iaq_init_continuous();
// sleep for the desired accelerated warmup time, e.g. 64s (See 3.9)
sleep(64);

// enter measurement loop
while (1) {
    sgp_measure_tvoc_blocking_read(tvoc_ppb);
    sleep(30); // Measurement interval is 30s in ultra-low power mode, 2s in low power mode
}
```

3.15. 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);
```

3.16. 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
}
```

3.17. 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"

3.18. Call `sgp_get_tvoc_inceptive_baseline()` to retrieve the on-chip inceptive baseline and call `sgp_set_iaq_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_iaq_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
March 2017	1.2.0	3-5	Update baseline documentation
March 2017	1.4.0	3	Update baseline persistence documentation
April 2017	1.5.0	1, 3	SGP30
May 2017	2.0.0	all	Change interfaces from resistance to ethanol and h2 signals
May 2017	2.0.1	all	Document signal scaling
July 2017	2.1.0	all	SGPC3
September 2017	2.3.0	3, 4	Accelerated warm-up by iaq_init…()
March 2018	2.4.0	all	Remove confidentiality footer, document sgp_iaq_init_continuous, sgp_set_absolute_humidity, sgp_set_power_mode, sgp_measure_test, sgp_get_driver_version.
April 2018	2.5.0	5	Drop signal scaling
July 2018	2.6.0	7	Factory Baseline
August 2018	2.7.0	4	Update accelerated warm-up times
February 2019	2.7.1	7	Rename factory baseline to 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