Application Note for Liquid Flow Sensors

Implementation Guide to the I²C Protocol

Summary

This document describes the main aspects of the I²C communication with Sensirion's Liquid Flow Sensors and provides instructions on how to implement the protocol on a controller system (master) for the communication with a sensor as the I²C device (slave).

This document describes the general implementation of the I²C communication for the following liquid flow sensor series:

SLI, SLS, SLG, SLQ, LG16, LS32, LPG10

The implementation assumes a hardware I²C interface on the master device.

Table of Contents

1. How to use this Guide

After years of supporting many development projects based our liquid flow sensors we want to share the following recommendations and suggest to follow them at least partially:

General Suggestions:

- Make use of the Arduino examples available online (https://github.com/Sensirion/arduino-liquid-flow-snippets)
- Ideally, get an Arduino UNO with a bread board to be able to use the code examples. This gives you a reference to see what should happen when you perform a step. The Arduino/Genuino Starter Kit (is a great starting point.
- First, implement **only** the flow measurement as described in section 4.4. Do **not** proceed to more complex communication sequences before you get changing readings close to 0x0 or 0xFFFF[1] at zero flow or with an empty flow channel. Do not proceed if you get a constant reading of 0xFFFF for each measurement.
- Next you need to implement reading the EEPROM to read the scale factor and the measurement unit, see sections 0 and 6.1.
- Now you can proceed to implement other functionalities as needed.
- Be **extremely** careful about writing to the EEPROM as it can destroy the sensor. In almost all cases writing to the EEPROM must not be part of your product's/system's code. It should be placed in a separate tool that is only used during setup and installation; for example, to change the address or save your own data in the free EEPROM space.

⚠ **Do not work on any routines that write to the EEPROM after 5 pm or under the influence of jetlag ☺**

2. Hardware Setup

Communication between a master and the sensor uses the digital I²C-interface. The physical interface consists of two bus lines, namely a data line (SDA) and a clock line (SCL). For the detailed specifications of the I²C protocol refer to the document "I²C-bus specification and user manual" from NXP. (http://www.nxp.com/documents/user_manual/UM10204.pdf)

The standard bus clock frequency is 100kHz, with the maximum bus clock frequency being 400kHz.

2.1 Communication Hardware

Possible hardware setups are:

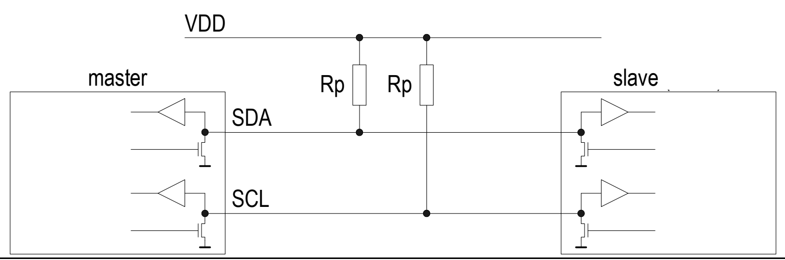
- Microcontroller(board) with integrated hardware I²C interface (like Arduino, …)
- Computer(board) with integrated hardware I²C interface (like Raspberry Pi, BeagleBone, …)
- PC with USB to I²C converter. (Note that the Sensirion USB-to-I²C Adapter Set has been designed for operation with the USB Sensor Viewer Software. It does not include an easy-to-use interface or software development kit)

[1] To understand this, see section 3.6 about the two's complement

2.2 Circuit Elements

Bi-directional bus lines are implemented by the devices (master and slave) using open-drain stages and a pull-up resistor connected to the positive supply voltage.

I²C External Circuit Elements



Both bus lines, SDA and SCL, are bi-directional and therefore require an external pull-up resistor.

The optimal pull-up resistor value is dependent on the system setup (capacitance of the circuit/cable, bus clock frequency). In most cases, 10 kΩ resistors are a reasonable choice. Connect the sensor's GND and the VDD lines to the supply voltage specified in your sensors datasheet.

⚠ The capacitive loads on the SDA and SCL lines have to be the same. It is important to avoid asymmetric capacitive loads.

⚠ Be sure to use the correct supply voltage (VDD) and communication levels for the given products.

⚠ Some microcontrollers have internal Pull Up resistors and might work without external ones; this does not mean that it is a stable design, especially with multiple sensors on one bus. Test your design.

2.3 Level Shifter

Some liquid flow sensors rely on 3.3V, others on 5V levels for the communication. Your hardware might need the communication to run on 3.3V (like the Raspberry Pi) or 5V (like the Arduino). If this does not match the product you are using, you will need to use level shifters between the sensor(s) and the microcontroller.

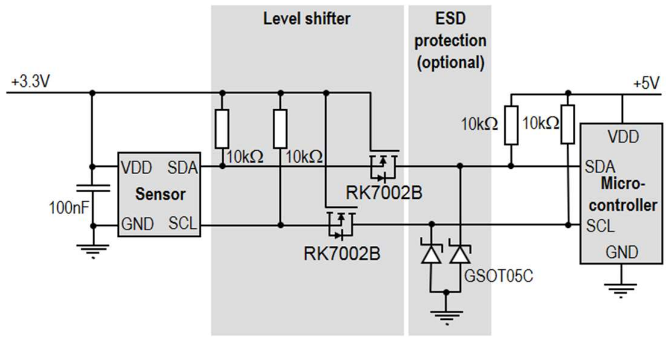
The following sensors have internal level shifters and can communicate at 3.3V and 5V:

LG16, LS32, SLI, SLS, SLG, SLQ-QT105

The following do not have internal level shifters:

LPG10, SLQ-QT500

Note: In some cases, a sensor with 3.3V levels might even work on a 5V bus. However, this is not recommended as it might shorten the lifetime of the sensor.



For more details on bidirectional level shifter for I^{2}C -bus please refer to US 5,689,196 and Application Note AN10441 by NXP: http://www.nxp.com/documents/application_note/AN10441.pdf.

2.4 Hard Reset

In all critical applications we recommend that you add a hard reset function that can be controlled by your system's master. The master must be able to interrupt the sensor's supply voltage and pull the SDA and SCL lines low to allow a clean reset of the sensor.

2.4.1 Hardware for Hard Reset

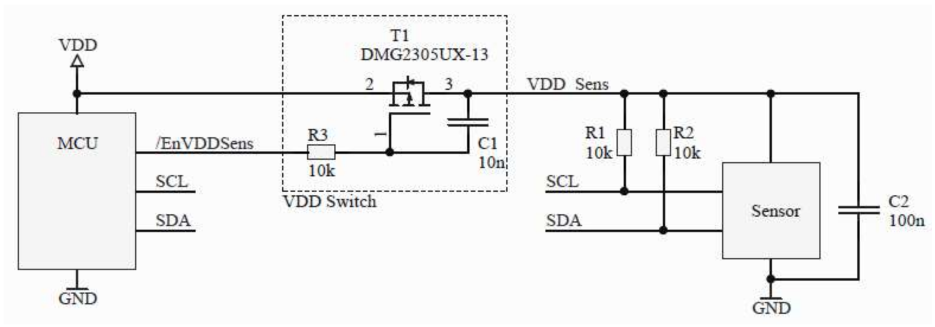
Details of the implementation depend on your system, the specific sensor and other requirements. However, below are three suggestions for common cases:

Voltage Regulator with Enable Pin

In the case that the sensor supply voltage is lower than the system voltage (for example: 5V microcontroller with the LPG10 at 3.3V) a possible solution is using a voltage regulator, e.g. the TCR2EF33 by Toshiba Semiconductor, with an enable pin that is controlled by the microcontroller.

P-Channel MOSFET

In a system with identical system and sensor voltage a regular P-Channel MOSFET, like the DMP2160UW, can be used (for example, when using an LG16 in a 5V system).



Care must be taken with sensors that have additional internal voltage regulators; like the LG16, SLI and SLG sensor. This can cause the system voltage to break down due to the high currents drawn at power-on. Adding a capacitor for stabilization can resolve this issue. This is not a problem for sensors, like the LPG10 and SLQ-QT500, without internal voltage regulator.

Load Switch

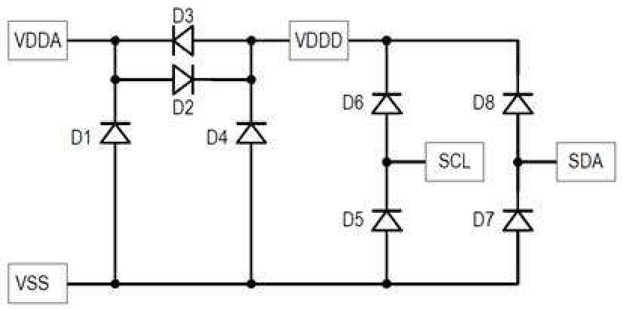
If the micro controller runs on a lower voltage than the sensor, an integrated load switch like the AP22802AW5-7 or TCK107AF can be used.

2.4.2 Reset Procedure

For a successful hard reset it is necessary that the SCL and SDA lines must be pulled low.

This is usually not a problem when working with open-collector pins (typical for I^{2}C); in this case it is sufficient to hook the pull-up resistors up to the sensor's VDD *after* the voltage regulator or MOSFET (see figure above).

In the case that the pull-up resistors remain connected to a high line during reset it is necessary that the master actively pulls down SCL and SDA because the sensor chip might remain powered through the diodes D6 and D8, on a level unsuitable for a clean reset. This is due to the sensor chip's internal ESD protection structure (see figure below):



The pins should remain low during power-off and for 31 ms after power-on to ensure a properly completed reset procedure.

Depending on your implementation of the I^{2}C interface you might have to use different methods to ensure that the lines are low during hard reset:

- If you have direct control, simply set the output pins low
- If you can trigger individual start and stop conditions, you can use the start condition to pull both lines low and the stop condition to pull them up again
- If you have no control over low level functions, you might have to disable I^{2}C to gain control of the output pins.

3. I²C Protocol

The I²C interface is a serial, half-duplex computer bus used for intra-board and short distance communication between microcotrollers and processors and peripheral ICs.

Many modern microcontrollers offer a hardware I²C interface. This means that it is not necessary to handle the low level I²C communication details in your code but you use the high level functions of the I²C module and let it do its work; possibly simultaneously while your micro controller performs other tasks. Below we provide the communication protocol for completeness. When using high level library functions for the communication, a detailed understanding of the base protocol is not necessary. However, it can still be helpful for trouble shooting communication issues.

3.1 Communication Protocol

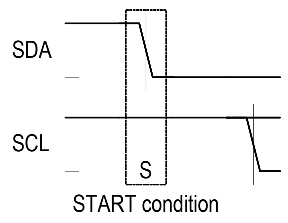
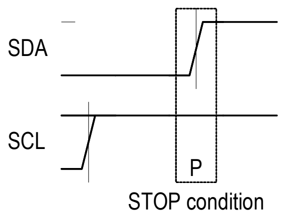
On the I²C bus all transactions begin with a START (S) and are terminated by a STOP (P) condition.

Transmission START Condition (S)

The START condition is a unique situation on the bus created by the master, indicating to the slaves the beginning of a transmission sequence (bus is considered busy after a START).

Transmission STOP Condition (P)

The STOP condition is a unique situation on the bus created by the master, indicating to the slaves the end of a transmission sequence (bus is considered free after a STOP).

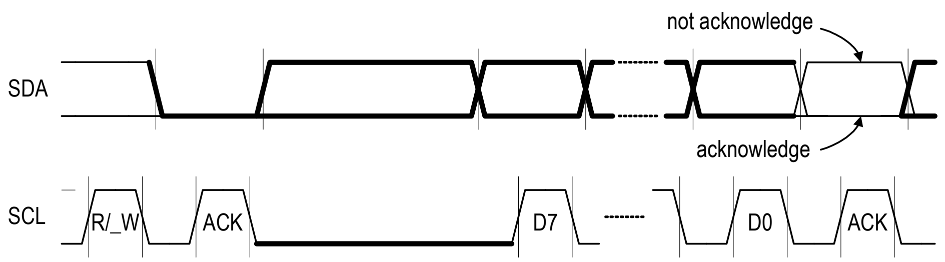
I²C Transmission Start Condition	*I²C Transmission Stop Condition*
 START condition	 STOP condition
A HIGH to LOW transition on the SDA line while SCL is HIGH	A LOW to HIGH transition on the SDA line while SCL is HIGH.

Acknowledge / Not Acknowledge

Each byte (8-bit) transmitted over the I²C bus is followed by an acknowledge pulse (ACK) from the receiver, e.g. for transmission from master to slave the pulse is generated by the slave.

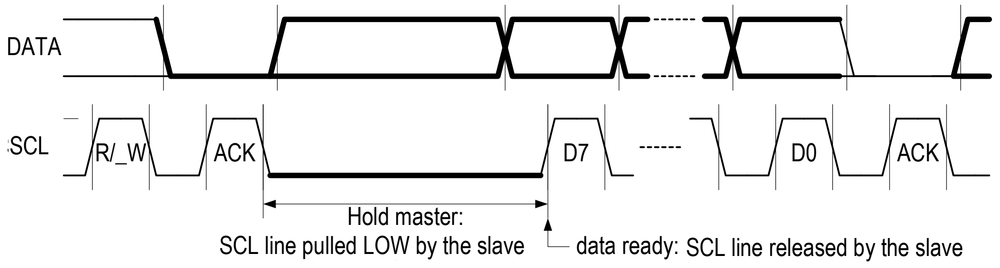
Transmission master to slave: in this case the slave generates an acknowledge pulse if it receives a valid command or data.

Transmission slave to master: the master generates an acknowledge pulse. If the acknowledge pulse is missing, the slave aborts the transmission of any following bytes and goes into idle mode.

I²C Acknowledge / Not Acknowledge
 not acknowledge SDA acknowledge SCL R/_W ACK D7 D0 ACK
Each byte is followed by an *acknowledge* or a *not acknowledge* pulse, generated by the receiver. (The bold lines indicate that the sensor controls the SDA/SCL line.)

3.2 Handshake Procedure (Hold Master, clock stretching)

In a master-slave system it is normally the master dictating when the slaves shall receive or transmit data. However, in some situations a slave device may need time to store received data or prepare data to be transmitted. Therefore, a handshake procedure is required allowing the slave to indicate termination of internal processing. While normally the master controls the clock line, in this case the slave exceptionally holds the clock line low and the bus is blocked until the slave releases the clock line.

I²C Hold Master
 DATA SCL R/_W ACK D7 D0 ACK Hold master: SCL line pulled LOW by the slave data ready: SCL line released by the slave
After the acknowledge-related SCL pulse, the sensor (slave) can pull down the SCL line to force the master into a wait state. By releasing the SCL line the sensor indicates that internal processing is terminated and that transmission can continue. (The bold lines indicate that the sensor controls the SDA/SCL line.)

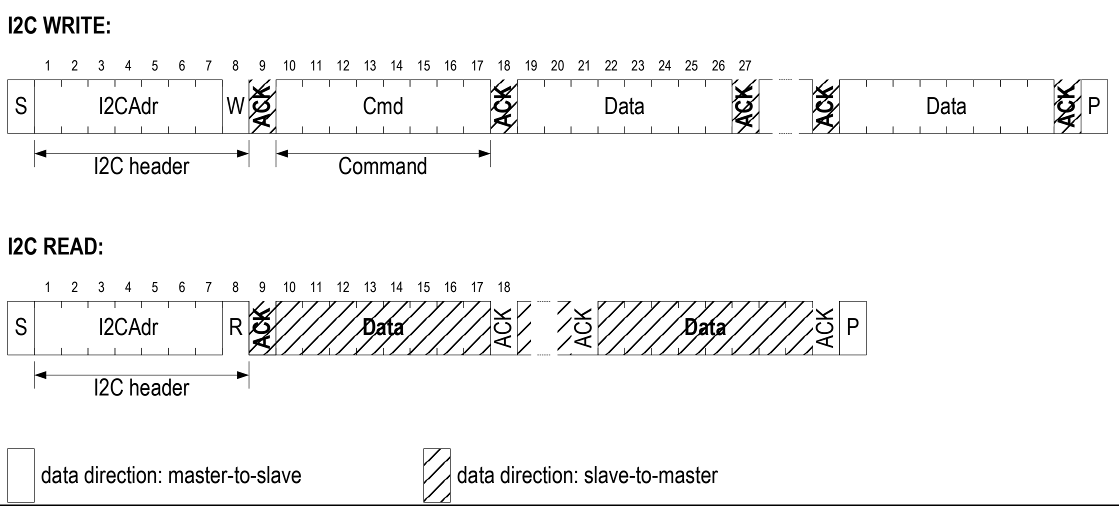
3.3 Data Transfer Format

In the I²C protocol, data is transferred in byte packages, i.e., in frames of 8-bit length. Each byte is followed by an acknowledge bit. Data is transferred with the most significant bit (MSB) first.

The data transfer format is shown in the following figures. A data transfer sequence is initiated by the master producing the START condition (S) and sending a header byte. The I²C header is made up of the 7-bit I²C device address and a data direction bit (R/_W).

The value of the R/_W bit in the header determines the data direction for the entire rest of the data transfer sequence: if R/_W = 0 (WRITE) the direction remains master-to-slave, if R/_W = 1 (READ) the direction changes to slave-to-master after the header byte.

I²C Read And Write



3.4 I²C Address

The I²C address consists of a 7-digit binary value. By default, the I²C address of Sensirion liquid flow sensors is set to decimal 64 (binary: 1000 000). The address is always followed by a write bit (0) or read bit (1). The default hexadecimal I²C header for read access to the sensor is therefore 0x81, for write access it is 0x80.

3.5 Data Types and Representation

The data in the frames is transmitted in big-endian order, i.e. Most-Significant Byte (MSB) first. Integers can be transmitted as signed or unsigned integers. The following types of integers are used:

Integer Type	Size	Range	Examples
unsigned, 16-bit (u16t)	2 Byte	0 … 2^{16}-1	VDD measurement
unsigned, 32-bit (u32t)	4 Byte	0 … 2^{32}-1	Serial number
signed, 16-bit (i16t)	2 Byte	-2^{15} … 2^{15}-1	Flow measurement

3.6 Two's Complement

Signed integers are represented according to the two's complement convention. This means that the *N*-bit binary representation of a negative number –*x* is the two's complement of that number's absolute value |-*x*|.

The following recipes may be used to obtain the binary representations of negative numbers and to reconstruct the numerical value from the binary representations, respectively.

Note that the two's complement is widely used in computing and used for signed number representation in almost all microprocessors. Consult your microprocessor's or programming language's manual if you can use the native integer format directly.

Find the N-bit signed integer representation m corresponding to a number x

```
if x<0:                             # if the number is negative
    m = (|x| ^ (2**N - 1)) + 1      # compute the two's complement of its absolute value
else:                               # else the number is positive
    m = x                           # no computation needed
```

Here the operator `'| |'` denotes the absolute value, '^' denotes the bit-wise XOR (exclusive OR), '**' denotes the power as in `2**3 = 8`.

Examples:

-7 as 8-bit signed integer:
```
(|-7| ^ (2**8-1)) + 1 = (7 ^ (256-1)) + 1 = (7 ^ 255) +1 = 248 + 1 = 249 = 0xF9
```
-7 as 16-bit signed integer:
```
(|-7| ^ (2**16-1)) + 1 = (7 ^ 65535) + 1 = 65528 + 1 = 65529 = 0xFFF9
```

Find the number x represented by the N-bit signed integer m

```
if m & 2**(N-1) == 2**(N-1):        # if the most-significant bit of m is '1', the number is negative.
     x = -((m ^ (2**N – 1)) + 1)    # compute the two's complement
else:                               # else the number is positive
     x = m                          # no computation needed
```

Here the operator '`&`' denotes the bit-wise AND, '^' denotes the bit-wise XOR (exclusive OR), '**' denotes the power as in `2**3 = 8`.

Examples:

Find the number represented by the 8-bit signed integer 0xF7:
```
m= 0xF7 = 247
247 & 2**7 == 2**7 : True
```
therefore: `x= -((247 ^ (2**8-1)) + 1) = -((247 ^ 255) + 1) = -(8 + 1) = -9`

Find the number represented by the 16-bit signed integer represented by the bytes [0xF7, 0x34]:
```
m = 0xF7 * 2**8 + 0x34 = 247 * 256 + 52 = 63232 + 52 = 63284
63284 & 2**15 == 2**15 : True
```
therefore: `x= -((63284 ^ 65535) + 1) = -(2251 + 1) = -2252`

3.7 Checksum

All Sensirion liquid flow sensors implement a slave to master checksum. We highly recommend implementing the checksum in your code and checking each answer from the sensor for the correct checksum. For this you simply calculate the checksum over the two data bytes and compare it with the following CRC byte.
Should there regularly be checksum errors in your communication it is advisable to raise an error and check your design for robustness.

Liquid flow sensors implement the CRC-8 standard based on the generator polynomial

x8 + x5 + x4 +1.

Specifically, the sensor implements the CRC-8 standard with following parameters[2]:

Name	Width	Polynom	Init	RefIn	RefOut	XorOut	Check[3]
CRC-8	8	0x131 ($x^8+x^5+x^4+1$)	0x00	false	false	0x00	0xA2

[2] More information may eventually still be found on this webpage:
http://www.sunshine2k.de/coding/javascript/crc/crc_js.html

[3] CRC for an ASCII input string "123456789" (i.e. 0x31, 0x32, 0x33, 0x34, ..., 0x39)

There are various possible implementations to calculate the checksum. For details see the separate application note available from Sensirion.
Here is one possible implementation in C that runs on Arduino and calculates the checksum for a two-byte data array:

```
#define POLYNOMIAL 0x131
byte data[2]; byte byteCtr; byte calc_crc = 0;

for (byteCtr = 0; byteCtr < 2; ++byteCtr){
  calc_crc ^= (data[byteCtr]);
  for ( i = 8; i > 0; --i){
    if (calc_crc & 0x80) { calc_crc = (calc_crc << 1) ^ POLYNOMIAL;}
    else { calc_crc = (calc_crc << 1); }
  }
}
```

3.8 Sensor commands

The following commands are available for I²C communication. The usage is explained in the following sections.

Command	Value	Description
USER_REG_W	0xE2	Write user register
USER_REG_R	0xE3	Read user register
ADV_USER_REG_W	0xE4	Write advanced user register
ADV_USER_REG_R	0xE5	Read advanced user register
TRIGGER_FLOW_MEASUREMENT	0xF1	Trigger a flow measurement
TRIGGER_TEMP_MEASUREMENT	0xF3	Trigger a temperature measurement
TRIGGER_VDD_MEASUREMENT	0xF5	Trigger a supply voltage measurement
EEPROM_W	0xFA	Write to EEPROM
EEPROM_R	0xFA	Read EEPROM
SOFT_RESET	0xFE	Soft reset

3.9 Control/Status Registers

All functions and options of the sensor are controlled by the contents of the control registers. The registers are initialized from the EEPROM at boot time and can be read and written to determine and configure the sensor's settings.

The available registers are:

Name	*Width [bit]*	*Description*
User Register	16	Basic user settings
Advanced User Register	16	Advanced user settings

Register settings can be changed in the registers themselves for immediate effect on the sensor's behaviour. The settings in the registers are volatile and will be initialized again at boot time.

The default boot content of the registers can be changed in the EEPROM. After changing the EEPROM entry, the sensor will read the new default contents from the EEPROM at every soft reset or hard reset.

In the final product/system the registers are typically only directly manipulated to alter the sensor's behavior during runtime, while the default boot content in the EEPROM is only changed once at production/installation.

For example, in a system that can handle multiple liquids, the user register is changed directly to set the active calibration field if the working liquid is changed during runtime. In contrast, the default boot content of the user register should be changed at installation if the system always uses the same active calibration.

See sections 5.2 and 5.3 for instructions on how to read and write the registers. See sections 5.4 and 0 for instructions on how to read and write the registers' default boot content in the EEPROM.

The default boot contents are stored at the following EEPROM addresses:

Wordadr	*#Words (16 bit)*	*Description/Format*
0x2C0	1	default boot content user register
0x2C1	1	default boot content advanced user register

To change a setting in a register, the following steps have to be performed:

1. Read the register to a variable in your program to determine the current state.
2. Change **ONLY** the bits of the variable that relate to the desired change.
3. Write the whole new 16-bit value to the register.
4. Reread the register to verify that the new settings have been applied correctly and no communication error has occurred.

To change the default boot content of a register, the following steps have to be performed:

1. Read EEPROM entry at the relevant EEPROM addresses to a variable in your program.
2. Change ONLY the bits of the variable that relate to the desired change.
3. Write the whole new 16-bit value to the EEPROM.
4. Reread the EEPROM entry to verify that the new settings have been applied correctly and no communication error has occurred.

Whenever writing to a register or to the EEPROM you must always write the **full word** (16 bit).

The calibration of the sensor is done for specific sensor settings. Changing the user and advanced user register may change the calibration and lead to non-interpretable sensor measurements. Overwriting the default boot content changes, the sensor settings irreversibly and may render the sensor unoperational. Sensirion disclaims any warranty for sensors with changed default boot content. Do not write to any other field of the EEPROM. This may destroy the configuration and calibration of the system.

The following tables contain a detailed description of the register contents.

3.9.1 User Register

When changing the register entry, read out the entire register (full 16-bit word) and do not change those settings labelled as "advanced settings" in the table below.

Bit	*#Bits*	*Description/Coding*
15:7	9	advanced settings (do not change)
6:4	3	selector active calibration field (i.e., active LUT (look-up-table)) **000: calibration field 0 (default)** 001: calibration field 1 010: calibration field 2 011: calibration field 3 100-111: calibration field 4 → *Utilization and content of calibration fields are dependent on the specific device. See your sensor's datasheet or contact Sensirion for further information.*
3:0	4	advanced settings (do not change)

3.9.2 Advanced User Register

When changing the register entry, read out the entire register (full 16-bit word) and do not change those settings labelled as "advanced settings" in the table below.

Bit	*#Bits*	*Description/Coding*
15:13	3	advanced settings (do not change)
12	1	no heater shut-down at measure end 0: shut-down heater 1: keep heater power as-is
11:9	3	resolution of measurement 000: 9 bit (flow); 6 bit (temperature/supply voltage) 001: 10 bit (flow); 7 bit (temperature/supply voltage) 010: 11 bit (flow); 8 bit (temperature/supply voltage) 011: 12 bit (flow); 9 bit (temperature/supply voltage) 100: 13 bit (flow); 10 bit (temperature/supply voltage) 101: 14 bit (flow); 11 bit (temperature/supply voltage) 110: 15 bit (flow); 12 bit (temperature/supply voltage) 111: 16 bit (flow); 13 bit (temperature/supply voltage) → *Default settings dependent on product.*
8:2	9	advanced settings (do not change)
1	1	hold-master 0: disabled **1: enabled (default)**
0	1	advanced settings (do not change)

4. Measurement

4.1 General

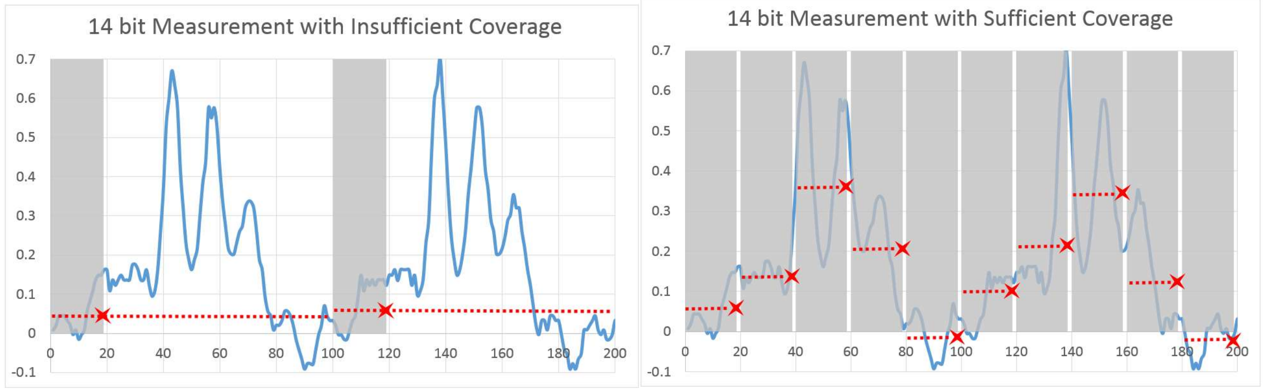
Flow measurements are always performed by triggering the measurement first and retrieving the measurement result once the measurement is finished.

A single flow measurement's duration depends on the resolution setting and can last between approximately 1 and 70 ms (see 0). Depending on this setting, the sensor will average multiple measurements internally for a higher resolution measurement.

It is important to understand that the measurement starts right after it is triggered by the I^{2}C read header and lasts only as long as the processing time (depending on the resolution setting). The next measurement only starts after it is triggered again. Therefore, irregular measurements will result in "blind spots" of the measurement. Since the communication only takes a few microseconds a nearly complete coverage is possible if the next measurement is triggered immediately.

In many cases, small delays between the measurements are acceptable. This can be needed if a fixed spacing between the measurements is desired. If, for example, 14 bit measurements with a typical duration of 17.5 ms are triggered every 20 ms the "blind spots" only make up 12.5 % of the measurement time; which will be acceptable in most cases.

In some applications only a low data rate of flow measurements is needed. It is important to understand that this is not the same as performing irregular flow measurements. Insufficient sampling rates of the flow profile can lead to incorrect measurement results due to aliasing effects, especially for pulsating flows (Nyquist theorem). If only low sampling rates are needed in the application the flow should still be sampled regularly but can be averaged over multiple measurements on the master.



Example: Insufficient sampling rate leads to incorrect results in case of pulsating flow. In the example on the left, one 14 bit measurement is started every 100 ms. The duration of the measurement is slightly less than 20 ms (grey area) and provides one measurement result for the average flow rate (red line) during this period (red star). Assuming that this measurement is valid for the remaining 80 ms results in a drastic underestimation of the flow rate (blue flow graph versus red line). In the example on the right, multiple measurements sufficiently cover the flow profile and can be averaged if desired.

In some system setups you might not have a dedicated microprocessor to handle the sensor measurement exclusively or want to measure/communicate with multiple sensors/devices on the same bus. In this case it might be a good approach to use the sensor without hold master (see 4.5 and 6.5.)

4.2 Processing Time for Digital Output

The following table summarizes the required processing time for all available resolutions. The times given do not include communication.

Resolution [bit]	*Processing Time [ms]*		
	Min.	*Typ.*	*Max.*
9	0.5	0.8	0.9
10	1.0	1.3	1.5
11	2.0	2.4	2.6
12	4.1	4.6	4.9
13	8.2	8.9	9.4
14	16.4	17.5	18.5
15	32.8	34.8	36.7
16	65.5	69.3	73.2

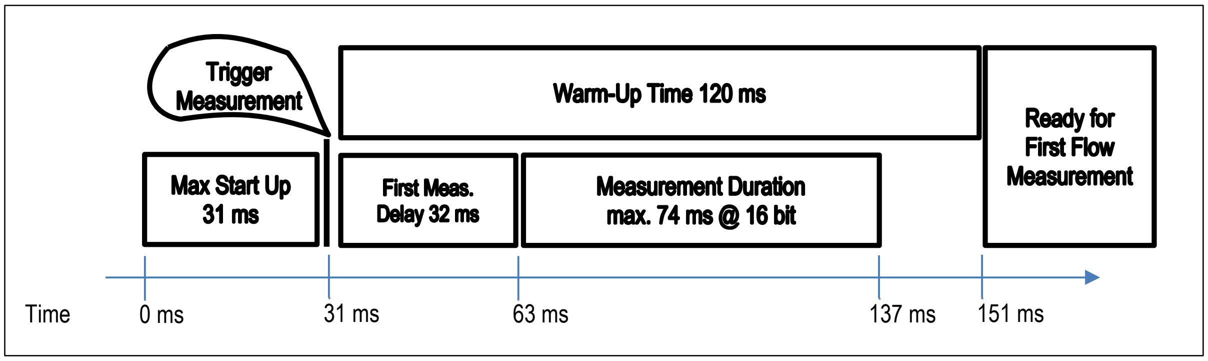
4.3 Sensor Start-Up and Heater Warm-Up Delay

When powering up the sensor, the SCL line should be HIGH (released) for the sensor to start-up. The maximum time for system power up is 2.7 ms until the sensor responds to communication requests. If the SCL line is pulled LOW the start-up can require a maximum of up to 31 ms.

The maximum time for a soft reset is 2.6 ms if the SCL line is HIGH and up to 31 ms if it is pulled LOW.

After reset or start-up of the sensor, the sensor's internal heater is turned off and must be started by performing a single (dummy[4]) measurement of the flow at any resolution setting. The very first measurement is delayed by approximately 32 ms for all liquid flow sensors, with the exception of the SLQ-QTxxx series, which performs the first measurement without any delay.

Due to the thermal measurement principle a total warm-up time of typically 120 ms is necessary for a reliable measurement. This includes the 32 ms needed for the heater warm-up delay and the time needed for the very first (dummy) measurement (e.g. 74 ms at 16 bit resolution).



[4] The term dummy measurement refers to the fact that the result of the measurement should be disposed of and does not serve as a regular measurement. The communication sequence does, however, not differ from any regular flow measurement as described in section 4.4.

4.4 Perform Measurement (Hold-Master Enabled)

To perform a measurement, first ***write the command byte 0xF1 (trigger measurement)*** to the sensor. In a second step ***perform a read operation*** to trigger the measurement and retrieve the flow measurement.

Upon reception of the I²C header with R/_W=1 (read data command), the sensor produces a hold-master condition on the bus (clock line is kept low) until the measurement is completed (duration depending on the configuration 1 to 80 ms, up to 112 ms (including heater warm-up delay for the very first measurement).

Make sure the I²C master is ready to wait and no communication with other devices on the same I2C bus is needed during this period because the master-hold condition blocks the entire I²C bus.

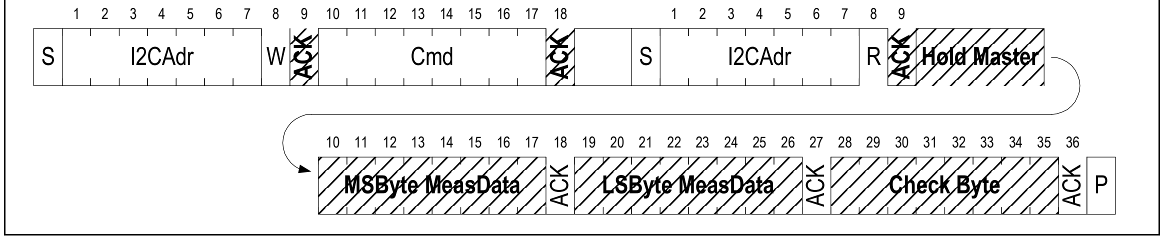
Once the hold-master condition is suspended, the master can read the result as two consecutive bytes which represent the current flow rate.

A CRC byte follows if the master continues clocking the SCK line after the second result byte. The sensor system checks whether the master sends an acknowledge after each byte and aborts the transmission if not.

Note that two transfer sequences are needed. The first sequence stores the command, while the second sequence invokes the previously stored command in READ mode.

Although, a subsequent I²C header with R/_W=1 will, by itself, start a new measurement after reading the data, for stability, we recommend to always resend the measurement command 0xF1 followed by the I²C header with R/_W=1 (read header) to start the next measurement.

I²C Measurement (hold-master enabled)



4.5 Perform Measurement (Hold-Master Disabled)

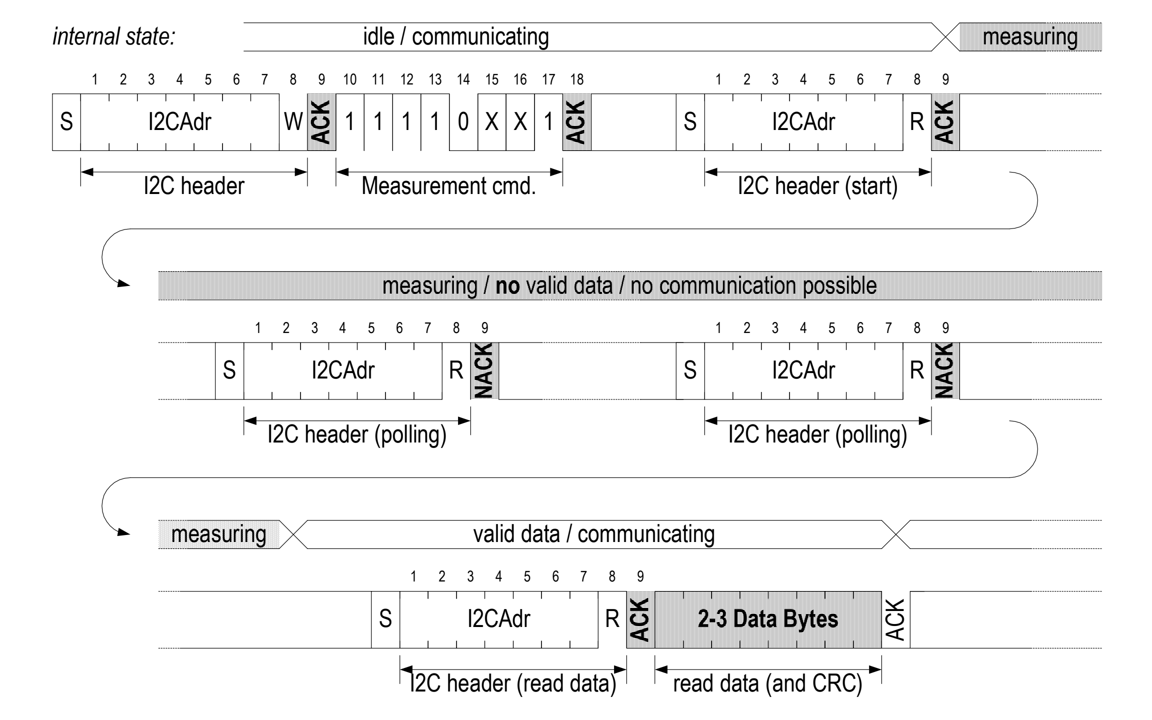
See sections 6.5 for instructions on how to disable the hold-master condition for the sensor.

When hold-master is disabled, send the flow measurement command 0xF1 followed by an I²C header with R/_W=1 (read header). The sensor starts measuring when it receives the read header. This is similar to the case that hold-master is enabled, but no hold-master condition is produced, the clock-line is released and the I²C bus is free. Send a second I2C read header to retrieve the measurement result from the sensor.

Until the sensor system has completed its measurement, an I²C header with R/_W=1 (read header) won't be acknowledged. Once the sensor has finished measuring, it will acknowledge an I²C read header and the master can read the result from the two consecutive bytes after the ACK. A CRC byte follows if the master continues clocking the SCL line after the second result byte (it is highly recommended to check to CRC byte).

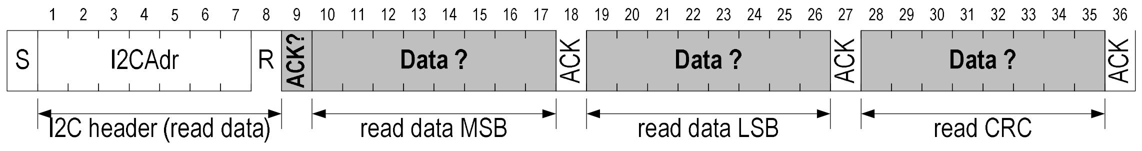
Although, a subsequent I²C header with R/_W=1 will, by itself, start a new measurement after reading the data, for stability, we recommend to always resend the measurement command 0xF1 followed by the I²C header with R/_W=1 (read header) to start the next measurement.

I²C, Measurement (hold-master disabled)



The following table shows the combinations of acknowledge or not acknowledged (NACK) pulse and the data the sensor system produces upon reception of an I²C read header. For all cases, the master sends an I²C header with R/_W=1 and reads three bytes, each of them followed by an acknowledge pulse generated by the master.

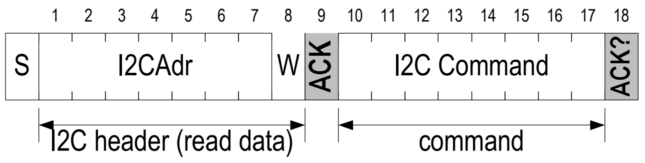
Hold-master disabled: possible responses to I²C read header



Meaning of I²C header	ACK/NACK (Bit 9)	Data MSB	Data LSB	CRC	CRC valid?
start measurement	ACK	255	255	255	no
polling	NACK	255	255	255	no
read data	ACK	MSB	LSB	CRC	yes

An I²C **header** with R/_W=0 will always be acknowledged, regardless of the sensor system's internal state. However, no command (with the exception of 0xF6[5]) will be acknowledged during the measurement and until the result of the last measurement has been read. After the measurement has been retrieved a new command can be sent to the sensor.

[5] The command 0xF6 is a specific command of the sensor chips command set. It is not applicable to liquid flow sensors. Do not send the command 0xF6* to any liquid flow sensor. Two read headers are necessary after this command to recover.

Hold-master disabled: possible responses to I²C write header			
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 S I2CAdr W ACK I2C Command ACK? I2C header (read data) command			
State	Command	1st ACK (Bit 9)	2nd ACK (Bit 18)
Measuring / Measurement not read	All commands	ACK	NACK
Measurement read / idle	All valid commands	ACK	ACK

4.5.1 Examples for the Communication (Hold-Master Disabled)

Example 1 (Hold-Master Disabled): correct usage

Action (w.r.t. I²C master)	ACK?	Sensor response (hex)	Comment
send data: 0xF1	ACK		set up flow measurement
read 3 bytes	ACK	FF FF FF	flow measurement started
wait			
read 3 bytes	ACK	flow. meas.	flow value with correct CRC
chip is idle and ready for communication			

Example 2 (Hold-Master Disabled): writing instead of reading flow value

Action (w.r.t. I²C master)	ACK?	Sensor response (hex)	Comment
send data: 0xF1	ACK		set up flow measurement
read 3 bytes	ACK	FF FF FF	flow measurement started
send data: 0xE3	NACK		communication fails, chip expects a read header
send data: 0xE3	NACK		communication fails, chip expects a read header
wait			
read 3 bytes	ACK	flow. meas.	flow value with correct CRC
send data: 0xE3	ACK		command accepted
read 3 bytes	ACK	0E 00 6D	user register with correct CRC

Example 3 (Hold-Master Disabled): polling

Action (w.r.t. I²C master)	ACK?	Sensor response (hex)	Comment
send data: 0xF1	ACK		set up flow measurement
read 3 bytes	ACK	FF FF FF	flow measurement started
read 3 bytes	ACK	FF FF FF	measurement not yet finished (polling)
read 3 bytes	ACK	flow. meas.	flow value with correct CRC
read 3 bytes	ACK	FF FF FF	flow measurement started
read 3 bytes	ACK	flow. meas.	flow value with correct CRC
read 3 bytes	ACK	FF FF FF	flow measurement started

send data: 0xE3	NACK		communication fails, chip expects a read header
read 3 bytes	ACK	flow. meas.	flow value with correct CRC
send data: 0xE3	ACK		command accepted
read 3 bytes	ACK	0E 00 6D	user register with correct CRC

4.6 Read Temperature and Supply Voltage

It is possible to read out the temperature and the supply voltage of the sensor chip. The communication sequence is identical to the flow measurement, described in sections 4.4 and 0. Instead of the command 0xF1 for a flow measurement the following commands must be send:

8-Bit Command Code	*Command*
0xF3	trigger temperature measurement
0xF5	trigger VDD measurement

Parameter	*Scale factor*	*Explanation*
Temperature	10°C^{-1}	Divide the signed output value by 10 to get the physical value in degree C.
Supply voltage	1mV (1000 V^{-1})	The output directly represents mV.

Attention: Temperature and VDD might not be calibrated for your product. Please consult the datasheet for specification.

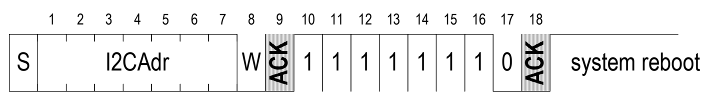
5. Transfer Sequences

5.1 Soft Reset (0xFE)

Force a reboot of the sensor system without switching the power off and on again. Upon reception of this command the sensor system reinitializes the control/status register contents from the EEPROM and starts operation according to the settings.

See sections 4.3 for timing details. The soft reset is only available when the sensor is idle.

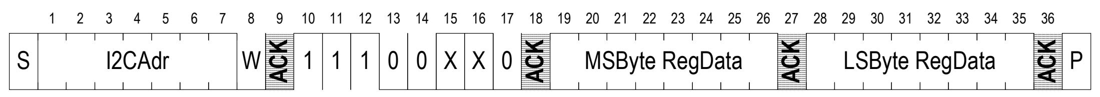
I²C, Soft Reset



5.2 Write Register (0xE2, 0xE4)

Overwrite the register addressed by the command. After receiving the write header with R/_W=0, followed by the respective command, the sensor system reads the new register value from the bus. The first byte is stored as the most significant byte, the second byte is stored as the least significant byte of the register. The sensor system acknowledges successful reception of each byte (ACK)

I²C, Write Register

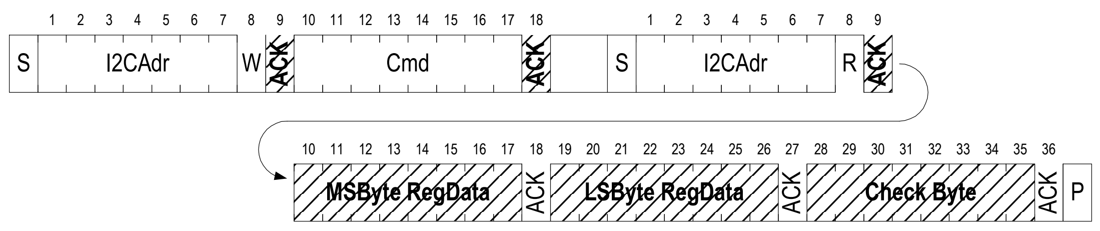


Because the sensor does not implement a master to slave checksum we strongly recommend to read back the register after every write operation to confirm that the correct values have been received by the sensor.

5.3 Read Register (0xE3, 0xE5)

Read the content of the register addressed by the command. After receiving the read the header with R/_W=1, the sensor system writes the register value to the bus. The first byte written is the most significant byte, the second byte the least significant byte of the register. A CRC byte follows, if the master continues clocking the SCL line after the second byte. The sensor system checks whether the master sends an acknowledge after each byte and aborts the transmission if not. Note that two transfer sequences are needed. The first sequence stores the command, while the second sequence invokes the previously stored command in READ mode.

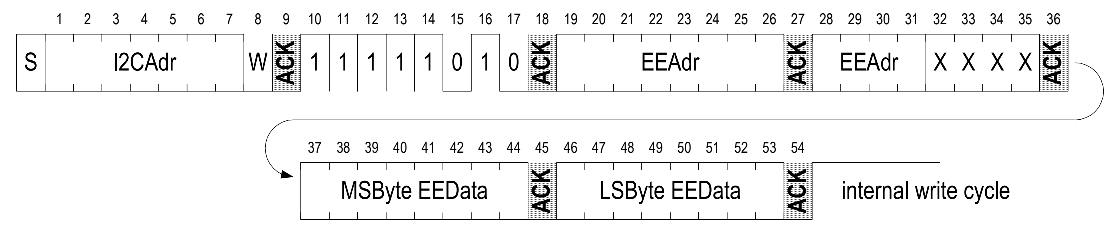
I²C Read Register



5.4 Write EEPROM (0xFA)

Overwrite the EEPROM memory cell (16-bit word) at the address provided. The first two bytes after the write header are interpreted as EEPROM address, beginning with the most significant bit (the four least significant bits of the second byte are ignored, since the address width is 12-bit only). After the address bytes, the new word value is read from the bus, most significant byte first. The sensor system acknowledges successful reception of each byte (ACK). Upon successful reception of the least significant byte the internal write cycle is triggered. The sensor system will not respond to further bus calls until the write cycle is completed.

I²C, Write EEPROM



⚠ Because the sensor does not implement a master to slave checksum we strongly recommend to read back the written EEPROM cells in order to verify correct write operation.

⚠ Care must be taken when constructing the two address bytes. Since the address is only 12 bit long it must be padded with 4 unused bits on the right. Example: When setting the address to 0x123, you must send 0x1230

EEPROM cells can be overwritten thousands of times, but it is advised to limit the writing to the EEPROM to the minimum.

5.4.1 General Conditions for Write EEPROM Operations

Additionally to the conditions given in datasheet, the following conditions have to be respected when writing to the EEPROM:

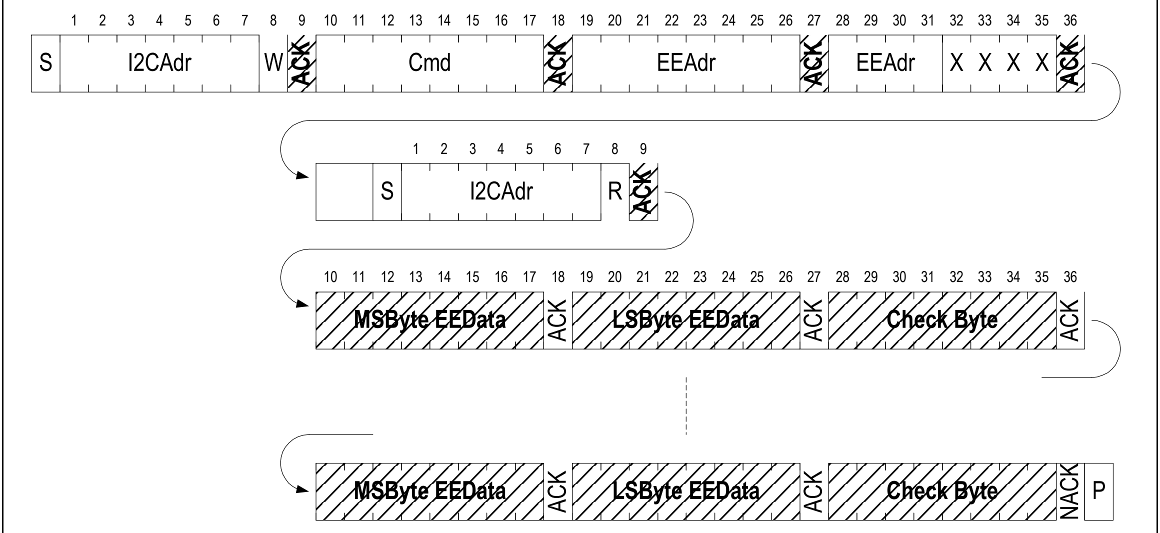
Property	*Min.*	*Typ.*	*Max.*	*Unit*
Bus Clock Frequency	0	100	400	kHz
EEPROM Internal Write Cycle		6	10	ms

5.5 Read EEPROM (0xFA)

Dump the content of the EEPROM starting at the address stored in the internal address register. The internal address register must be set by invoking a dummy write sequence, i.e., initiating a write EEPROM sequence by sending the command 0xFA and the two address bytes.

After the read header with R/_W=1, the sensor system writes the EEPROM word at the current address to the bus, followed by a CRC byte. The internal address register is automatically increased after each completed word write out. Therefore, if the master keeps on clocking the SCL line after the CRC byte, the next EEPROM word with corresponding CRC byte is written to the bus, and so on. When the address pointer reaches the end of the address space it restarts at zero. The master can interrupt EEPROM dumping by not pulling down the SDA line during any acknowledge-related SCL pulse (NACK).

I²C Read EEPROM



6. Communication Sequences

6.1 Read Scale Factor and Measurement Unit

For some standard products, the default scale factor and flow unit can be found in the datasheet. Nevertheless, we strongly recommend to readout the scale factor and flow unit from the sensor itself.

The original calibrated signal read from the sensor is an integer number which is signed (see 3.6) for bi-directional calibration fields and unsigned for uni-directional calibration fields. To convert the integer value to the original unit the sensor has been calibrated for, divide it by the scale factor of the sensor's active calibration field.

The EEPROM addresses for the scale factors and the measurement units of each active calibration field are given below.

Calibration Field	*Address of scale factor*	*Address of measurement unit*
0	0x2B6	0x2B7
1	0x5B6	0x5B7
2	0x8B6	0x8B7
3	0xBB6	0xBB7
4	0xEB6	0xEB7

6.1.1 Read Out Scale Factor and Measurement Unit from EEPROM

To read out the scale factor for a given calibration field the following steps have to be performed:

1. Set the EEPROM's internal address pointer to the respective address by sending an I^{2}C write-header followed by the EEPROM access command (0xFA) and the address from which to read. Full command string: 0x 80 FA XX X0. (hexadecimal, XX X is the respective EEPROM address, assuming the default I^{2}C address, see section 0). **Important Note:** the EEPROM address is 12 bit long and needs to be padded with zeros on the right. I.e. the address 0x**123** needs to be transmitted as 0x**12** 0x**3**0.
2. Read the EEPROM entry (1 word) and the CRC byte.

The scale factor is a 16-bit unsigned integer value. For the measurement unit, determine the corresponding unit according to the coding provided in section 0 below.

It is also possible to read the scale factor and the measurement unit in two consecutive reads cycles as the word address for each calibration field's measurement unit always follows right after the scale factor and the sensor chip automatically increases the internal address pointer after it has written the EEPROM word to the I2C bus.

6.1.2 Scale Factor

To calculate the flow rate in physical units as a float point value from the integer value read from the sensor, divide the integer value by the determined scale factor. Note that you have to read the corresponding scale factors for all the calibration fields you are using and always apply the respective scale factor depending on the active calibration field.

The scale factor's sole purpose is to allow an efficient transfer of the flow measurement value as a 16-bit integer. The scale factor is not a calibration in itself and it is not possible to change the calibration by simply using a different scale factor.

Scale factor and measurement unit are defined for each calibration field during calibration and cannot be changed later on.

6.1.3 Measurement Unit Coding

The number which has been read out of the EEPROM for the flow unit can be de-coded according to below system. Only relevant units, dimensions and time bases are listed.

Unit	nl/min	µl/min	ml/min	µl/sec	ml/h
Code	2115 (0x0843)	2116 (0x0844)	2117 (0x0845)	2100 (0x0834)	2133 (0x0855)

Scale factor and measurement unit are defined for each calibration field during calibration and cannot be changed later on.

6.2 Read Product Details

All Sensirion liquid flow sensors provide a part name and serial number for traceability.

The serial number and part name of the sensor can be read out from the EEPROM at the following addresses:

Word address	*#Words (16 bit)*	*Description*	*Format*
0x2E8-0x2F1	10	sensor/part name	ASCII formated character array
0x2F8-0x2F9	2	serial number product	32 bit unsigned integer

To read the part name and the serial number the following steps have to be performed:

1. Set the EEPROM's internal address pointer to the respective address by sending an I^{2}C write-header followed by the EEPROM access command (0xFA) and the address from which to read. Full command string: 0x 80 FA 2E 80. (hexadecimal, assuming the default I^{2}C address, see 0)
2. Read the EEPROM entry (1 word) and the CRC byte by sending an I^{2}C header with R/_W=1.
3. Repeat step 2 eleven times in total by continuing to clock the SCL line. The internal address pointer is automatically moved to the next word after each completed word write out.

The ten 16-bit words starting at address 0x2E8 are a 20 byte, ASCII formatted character array. The two 16-bit words starting at address 0x2F8 provide the serial number and are formatted as a 32-bit unsigned integer (most significant byte first).

6.3 Change Resolution

The total time per measurement is determined by the resolution setting, bits <11:9>, in the advanced user register. The active resolution of measurement setting can be changed in the advanced user register to effect all following measurements until the setting is changed again or the sensor is reset.

Alternatively, the setting can be changed in the default boot content of the advanced user register in the sensor's EEPROM to be in effect after the next system reboot.

Additionally, the default resolution at boot time can also be changed using the USB-RS485 Sensor Viewer with the SCC1-USB sensor cable. The USB-RS485 Sensor Viewer is available in the download center on the Sensirion liquid flow webpage.

Typically, the resolution setting is only changed in the register during evaluation and if more than one resolutions (measurement times) are needed during operation.

If the same resolution setting shall be used at any time during operation the default boot content of the advanced user can be changed in the EEPROM once during installation.

Bit number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default Advanced User register for liquid flow sensors[6]	x_{15}	x_{14}	x_{13}	x_{12}	1	1	1	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
	Do not change!				resolution of measurement			Do not change!								
Changed register for the desired resolution setting	x_{15}	x_{14}	x_{13}	x_{12}	0/1	0/1	0/1	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0

6.3.1 Change Active Resolution

To change the sensor's active resolution setting the flowing steps have to be performed. The detailed register read/write sequences are explained in sections 5.2 and 5.3.

1. Read the advanced user register and the CRC byte with command 0xE5.
2. Save the register entry to a variable in your program.
3. Change **ONLY** bits <11:9> of this variable to the setting for the new resolution (see sections 3.9.2).
4. Write the entire new 16-bit value to the sensor's register with command 0xE4.
5. Reread the register to verify that the new settings have been applied correctly and no communication error has occurred.
6. The new resolution setting is active immediately.

6.3.2 Change Default Resolution

To change the sensor's default resolution at boot time the following steps have the be performed to modify the setting in the default boot content of the advanced user register (see section 3.9.2). The detailed EEPROM read/write sequences are explained in sections 5.4 and 0.

1. Set the EEPROM's internal address pointer to address 0x2C1 by sending an I^{2}C write-header followed by the EEPROM access command (0xFA) and the address from which to read. Full command string: 0x 80 FA 2C 10. (hexadecimal, assuming the default I^{2}C address, see section 0).
2. Read the EEPROM entry (1 word) and the CRC byte by sending an I^{2}C header with R/_W=1.
3. Save the EEPROM entry to a variable in your program.
4. Change **ONLY** bits <11:9> of this variable to the setting for the new resolution (see section 3.9.2).
5. Reset the EEPROM's internal address register to address 0x2C1 by repeating step 1 (the internal pointer has been incremented by reading the word).
6. Write the new value to the sensor's EEPROM.
7. Wait for the internal write cycle of 10ms.
8. Reread the EEPROM entry by repeating steps 1 and 2 to verify that the new settings have been applied correctly and no communication error has occurred.
9. Soft reset the sensor by sending 0xFE.
10. Wait for soft reset (up to 31 ms. see section 4.3).
11. You can verify that the active resolution has been changed by reading the active state of the advanced user register as described above (6.3.1, steps 1 and 2).

EEPROM cells can be overwritten thousands of times, but it is advised to limit the writing to the EEPROM to the minimum.

[6] With the exception of the sensors of the SLQ-QTxxx series.

6.4 Change Calibration Field

Many liquid flow sensors hold two or more calibrations. Each calibration is stored in a separate calibration field. The active calibration field is determined by the selector setting, bits <6:4> of the user register. The active calibration field can be changed in the user register to effect all following measurements until it is changed again or the sensor is reset.

Alternatively, the default calibration field (i.e. the active calibration field at power up) can be changed in the default boot content of the user register in the sensor's EEPROM to be in effect permanently after the next system reboot.

Follow the instructions below and see section 5.2, or section 5.4 for changing the register directly or the default boot content respectively.

Additionally, the default active calibration field at boot time can also be changed using the USB-RS485 Sensor Viewer with the SCC1-USB sensor cable. The USB-RS485 Sensor Viewer is available in the download center on the Sensirion liquid flow webpage.

Typically, the active calibration field is changed in the register during evaluation and if more than one calibrations are needed during operation.

If the same calibration is to be used at any time during operation the default boot content of the user register can be changed in the EEPROM during installation (one time change).

Bit number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default User register	x_{15}	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	0	0	0	x_3	x_2	x_1	x_0
	Do not change!									selector active calibration field			Do not change!			
Changed user register for active calibration field 1	x_{15}	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	0	0	1	x_3	x_2	x_1	x_0

6.4.1 Change Active Calibration Field

To change the sensor's active calibration field, the flowing steps have to be performed. The detailed register read/write sequences are explained in sections 5.2 and 5.3.

1. Read the user register and the CRC byte with command 0xE3
2. Save the register entry to a variable in your program.
3. Set **ONLY** bits <6:4> of this variable to the setting for the new resolution (see section 3.9.1)
4. Write the entire new 16-bit value to the sensor's register with command 0xE2.
5. Reread the register to verify that the new settings have been applied correctly and no communication error has occurred.
6. The new calibration field is active immediately.

6.4.2 Change Default Calibration Field

To change the sensor's default resolution at boot time the following steps have the be performed to modify the setting in the default boot content of the user register (see section 3.9.1). The detailed EEPROM read/write sequences are explained in sections 5.4 and 0.

1. Set the EEPROM's internal address pointer to address 0x2C0 by sending an I²C write-header followed by the EEPROM access command (0xFA) and the address from which to read. Full command string: 0x 80 FA 2C 00. (hexadecimal, assuming the default I²C address, see 0)
2. Read the EEPROM entry (1 word) and the CRC byte by sending an I²C header with R/_W=1
3. Save the EEPROM entry to a variable in your program
4. Set **ONLY** bits <6:4> of this variable to the setting for the new resolution (see section 3.9.1)
5. Reset the EEPROM's internal address register to address 0x2C0 by repeating step 1 (the internal pointer has been incremented by reading the word).

6. Write the new value to the sensor's EEPROM.
7. Wait for the internal write cycle of 10ms.
8. Reread the EEPROM entry by repeating steps 1 and 2 to verify that the new settings have been applied correctly and no communication error has occurred.
9. Soft reset the sensor by sending 0xFE.
10. Wait for soft reset (up to 31 ms. see section 4.3).
11. You can verify that the active calibration field has been changed by reading the active state of the user register as described above (see 6.4.1, steps 1 and 2).

EEPROM cells can be overwritten thousands of times, but it is advised to limit the writing to the EEPROM to the minimum.

6.5 Enable/Disable Hold Master

By default, Sensirion liquid flow sensors perform a master hold state (SCL line is pulled to low) during the measurement. By releasing the SCL line, the sensor indicates that the measurement is done and it is ready to send the measured value. This guarantees that the master is notified as soon as the measuremement is done. This blocks the bus for communication to other sensors during the measurement process.

There may be applications, where a free bus is required during the measurement for communication to other I^{2}C devices. Another reason to disable the hold master state can be power preservation for battery opperated applications. Without hold master, the sensor does not hold down the SCL line and therefore no energy is dissipated at the pull-up resistor.

By clearing bit 1 in the advanced user register, the so called "hold-master" may be disabled. The register can either be changed directly to effect all following measurements until it is changed again or the sensor is reset. Alternatively, the setting can be changed in the default boot content of the advanced user register in the sensor's EEPROM to be in effect permanently after the next system reboot.

Follow the instructions below and see sections 5.2 and 5.4 for changing the register directly or the default boot content respectively.

Typically, this setting is only changed in the register during evaluation, development and debugging.

Bit number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default advanced user register, hold-master enabled	x_{15}	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	**1**	x_0
	Do not change!														hold-master	Do not change!
Changed register to disable the hold-master	x_{15}	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	**0**	x_0

6.5.1 Change Active Hold Master Setting

To change the sensor's active hold master setting the flowing steps have to be performed. The detailed register read/write sequences are explained in sections 5.2 and 5.3.

1. Read the advanced user register and the CRC byte with command 0xE5
2. Save the register entry to a variable in your program.
3. Set/clear **ONLY** bit 1 of this variable to the new setting
4. Write the entire new 16-bit value to the sensor's register with command 0xE4.
5. Reread the register to verify that the new settings have been applied correctly and no communication error has occurred.
6. The new setting is active immediately.

6.5.2 Change Default Hold Master Setting

To change the sensor's default hold master setting at boot time the following steps have the be performed to modify the setting in the default boot content of the advanced user register (see 3.9.2). The detailed EEPROM read/write sequences are explained in sections 5.4 and 0.

1. Set the EEPROM's internal address pointer to address 0x2C1 by sending an I²C write-header followed by the EEPROM access command (0xFA) and the address from which to read. Full command string: 0x 80 FA 2C 10. (hexadecimal, assuming the default I²C address, see 0)
2. Read the EEPROM entry (1 word) and the CRC byte by sending an I²C header with R/_W=1
3. Save the EEPROM entry to a variable in your program
4. Set/Clear **ONLY** bit 1 of this variable to the setting for the new resolution (see section 3.9.2)
5. Reset the EEPROM's internal address register to address 0x2C1 by repeating step 1 (the internal pointer has been incremented by reading the word).
6. Write the new value to the sensor's EEPROM.
7. Wait for the internal write cycle of 10ms.
8. Reread the EEPROM entry by repeating steps 1 and 2 to verify that the new settings have been applied correctly and no communication error has occurred.
9. Soft reset the sensor by sending 0xFE.
10. Wait for soft reset (up to 31 ms. see section 4.3).
11. You can verify that the hold master setting has been changed by reading the active state of the advanced user register as described above (see 6.5.1, steps 1 and 2).

EEPROM cells can be overwritten thousands of times, but it is advised to limit the writing to the EEPROM to the minimum.

6.6 Heater Control

All Sensirion liquid flow sensors use the microthermal flow measurement principle. A micro-heater, integrated on the sensor's CMOSens® chip, introduces a tiny amount of heat into the liquid. The flow dependent heat profile is picked up by the nearby temperature sensors and their signals are used to measure the liquid's flow rate and direction.

In most applications it is not necessary to actively control the heater's state. It is sufficient to keep the heater turned on at all times and, possibly, account for the fact that the heater is being turned on with the first flow measurement. If you have reasons to believe that active heater control might be necessary in your application, contact Sensirion for support.

The behaviour of the micro-heater is controlled by bit 12 of the advanced user register. If the bit is set, the heater is kept powered up after a measurement. If the bit is cleared the heater is powered down after a measurement.

Bit number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default advanced user register, heater turned on	x_{15}	x_{14}	x_{13}	1	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
	Do not change!			heater setting	Do not change!											
Changed register to power down the heater after the next measurement	x_{15}	x_{14}	x_{13}	0	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0

Because bit 12 controls the behaviour of the heater's power **after** a measurement, it is necessary to follow any change of the register setting by a dummy measurement to switch the heater's power on or off.

If the regular measurement is performed at a high resolution setting and therefore takes a longer time, the active resolution could be changed to 9 bit to achieve the shortest possible duration of the dummy measurement.

6.6.1 Heater Mode On

To set the heater mode to permanently on, the following steps have to be performed:

1. Read the advanced user register and the CRC byte with command 0xE5.
2. Save the register entry to a variable in your program.
3. Set **ONLY** bit 12 of the variable to 1.
4. Write the entire new 16-bit value to the register with command 0xE4.
5. Reread the register to verify that the new settings have been applied correctly and no communication error has occurred.
6. Perform one dummy measurement with command 0xF1 to apply the setting and turn the heater on.

6.6.2 Heater Mode Off

To set the heater mode permanently off, the following steps have to be performed:

1. Read the advanced user register and the CRC byte with command 0xE5.
2. Save the register entry to a variable in your program.
3. Clear **ONLY** bit 12 of the variable to 0.
4. Write the new 16-bit value to the register with command 0xE4.
5. Reread the register to verify that the new settings have been applied correctly and no communication error has occurred.
6. Perform one dummy measurement with command 0xF1 to apply the setting and turn the heater off.

6.7 Change I^{2}C Address

The sensor's I^{2}C device address is stored in the sensor chip's EEPROM. By default, this address is preset to a fix address (0x40). It might be needed to change this address in order to use the sensor with other I^{2}C components on the same bus. To change the I^{2}C address, it is required to overwrite the default content in the EEPROM and reboot the sensor. There is no option to change the I^{2}C address in a (volatile) register.

The following steps have to be performed:

1. Set the EEPROM's internal address pointer to address 0x2C2 by sending an I^{2}C write-header followed by the EEPROM access command (0xFA) and the address from which to read. Full command string: 0x 80 FA 2C 20. (hexadecimal, assuming the default I^{2}C address, see section 0)
2. Read the EEPROM entry (1 word) and the CRC byte by sending an I^{2}C header with R/_W=1.
3. Save the EEPROM entry to a variable in your program
4. Define the new EEPROM entry according to the desired setting. (see below)
5. Reset the EEPROM's internal address register to address 0x2C2 by repeating step 1 (the internal pointer has been incremented by reading the word).
6. Write the new value to the EEPROM at the same address. (see 5.4 and example 6.7.2)
7. Wait 10ms for the internal write cycle to complete.
8. Reread the EEPROM entry by repeating steps 1 and 2 to verify that the new settings have been applied correctly and no communication error has occurred.
9. Soft reset the sensor by sending 0xFE.
10. Wait for soft reset. (up to 31 ms., see section 4.3)
11. Verify that the I^{2}C device address has been changed by communicating with the sensor at the new address.

EEPROM cells can be overwritten thousands of times, but it is advised to limit the writing to the EEPROM to the minimum.

Bit number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default EEPROM Word (16 bit) at 0x2C2	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	1	0	0	0	0	0	0	X_2	X_1	X_0
	Do not change!						I²C device address of sensor chip							Do not change!		
Changed Address (Example 0x41)	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	1	0	0	0	0	0	1	X_2	X_1	X_0

After writing to the EEPROM it is not possible to reset to the factory settings. Do not write to any other field of the EEPROM. This may destroy the configuration and calibration of the system. Sensirion disclaims any warranty for sensors with changed EEPROM entries other than the I²C address.

6.7.1 Reserved I²C device addresses

The I²C address range for devices with a seven-bit address is 0x0 through 0x7F. However, this range is not entirely available. Device address 0x0 is blocked in the sensor itself. On top of that, addresses 0x0 through 0x7 and 0x78 through 0x7F are special and reserved addresses in the I²C protocol. It is strongly advised to choose an address in the range from 0x8 through 0x77 when changing the device address.

6.7.2 Example Communication Sequence to Change the I²C Address

In this example we change the I²C device address from 0x40 to 0x21 in ten steps.

Step	*Master*	*Slave*	*Description*
1	0x80 FA 2C 20	(4 x ACK)	Master addresses the sensor and sets the pointer to EEPROM address 0x2C2
2	0x81	(1 x ACK)	Master addresses the sensor again, but now to read from the Sensor
3	(3 x ACK)	02 07 CRC	Sensor returns the EEPROM data of address 0x2C2 and a CRC byte (note that device address at bit <9:3> is 0x40 and that bit <2:0> are b111). Do not continue to the next step if unexpected values are read!
4	0x80 FA 2C 20	(4 x ACK)	Master addresses the sensor again and sets the pointer to EEPROM address 0x2C2
5	0x01 0F	(2 x ACK)	Master writes new device address (0x21) to bit <9:3> and b111 to bit <2:0> into the EEPROM
6	-	-	Wait for the internal write cycle of 10ms
7	0x80 FA 2C 20 81 (3 x ACK)	(4 x ACK) (1 x ACK) 01 0F CRC	Repeat step 1-3 and check EEPROM data of address 0x2C2. Note that the sensor still has to be addressed with the old address since the device has not been powered down or reset. However, EEPROM data has been successfully changed to 0x01 0F.
8	0x80 FE	(2 x ACK)	Soft reset. Power down is also possible.
9	0x80	(1 x NACK)	Master checks if the I²C device address is changed by addressing the sensor with the old address. The master receives a NACK
10	0x42	(1 x ACK)	Master addresses the sensor with address 0x21 and sensor responds with an ACK. The I²C device address has been changed successfully

6.8 Access Free EEPROM Space

Calibration data and all settings of the sensor are stored in the embedded EEPROM of the sensor chip. 31 words of 16 bit in the EEPROM are free to be accessed by the user. The rest of the EEPROM must not be overwritten, because the configuration and calibration of the sensor could be affected and the sensor might become irreversibly damaged.

Wordadr	*#Words (16 bit)*	*Description/Format*
0xFE0 - 0xFFE	31	Free space in EEPROM

To access the free space in the EEPROM, the following steps have to be performed:

1. Set the EEPROM's internal address pointer to respective address by sending an I²C write-header followed by the EEPROM access command (0xFA) and the address from which to read. Full command string: 0x 80 FA XX X0. (hexadecimal, assuming the default I²C address, see section 0 and 0xXXX is the desired address within the free EEPROM space.)
2. Read the EEPROM entry (1 word) and the CRC byte.
3. Define the new EEPROM entry according to the desired content.
4. Reset the EEPROM's internal address register to the desired address by repeating step 1 (the internal pointer has been incremented by reading the word).
5. Write the new value to the relevant EEPROM address.
6. Reread the EEPROM entry at the relevant EEPROM address to confirm that the correct data has been written.

Steps 1 and 2 can be skipped if the entire word shall be overwritten completely.

⚠ **The calibration of the sensor is done for specific sensor settings. Changing the EEPROM content may change the calibration and lead to non-interpretable sensor measurements. Overwriting the default boot content changes, the sensor settings irreversibly and may turn the sensor useless. Sensirion disclaims any warranty for sensors with changed EEPROM content. Do not write to any other field than the free space of the EEPROM. This may destroy the configuration and calibration of the system.**

7. Error Handling

We strongly recommend implementing handling procedures for the possible error cases outlined below.

7.1 Sensor to Master Data Corruption (Checksum Error)

When reading measurement data, register or EEPROM content, check the CRC byte after every two bytes of data (as described in section 3.7). If incorrect, repeat the measurement or reread the register/EEPROM.

Possible causes for checksum errors include

- incorrect implementation of the checksum computation on the master
- overlapping commands on the communication bus. For instance, if the master sends the next command before the reply to the previous command has arrived, the reply from the slave may overlap with that next command sent by the master
- several devices on the bus have the same address and their replies to a command overlap
- communication is disturbed by electrical interference from very harsh electromagnetic environments.

7.2 Master to Sensor Data Corruption

It is also possible that data that is sent from the master to the sensor is corrupted in any conceivable way. Since the sensor does not implement a master to sensor checksum, a corrupted command or value is not detected automatically. Therefore, it is recommended to read back the register and EEPROM after writing a change.

Even though it is possible to start a new measurement with a read header (which is not acknowledged by the sensor), it is advisable to send the command 0xF1 to trigger the flow measurement before each start of a new measurement because corrupt communication could potentially be interpreted by the sensor as a different command and cause the sensor to leave the measurement mode.

7.3 Sensor Timeout

We recommend to implement a timeout for all communication with the sensor. Should the sensor not respond after the chosen timeout time, the sensor can either be hard reset immediately or after it does not respond to a test command (e.g. 0xF1).

If the sensor does not end the hold master condition and blocks the bus indefinitely after a measurement has been started, the only solution is to perform a hard reset.

Note: A frozen sensor is very unlikely and most customers will never experience it in their application. However, implementing the timeout and hard reset (as described in section 2.4) is strongly recommended for all applications that cannot accept a full system reset in the case of a sensor freeze.

7.4 SDA High after Stop Condition

Interference on the communication can cause the sensor to miss a Stop Condition issued by the master. Therefore, we suggest to check that the SDA line is HIGH after the Stop Condition. In case of a failure, the sensor still wants to send data and pulls down the SDA line.

To resolve this failure case, the master must clock the SCL line 9 more times while keeping the SDA line high. With a I^{2}C library this can be realized by issuing a read without ACK.

Alternatively, a hard reset will also resolve this failure case and might be necessary if clocking SCL does not help.

Revision History

Date	Version	Changes
October 2017	1	First Version

Headquarter and Subsidiaries