Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

ENS210

Relative Humidity and Temperature Sensor with I²C Interface

General Description

The ENS210 integrates one relative humidity sensor and one high-accuracy temperature sensor. The device is encapsulated in a QFN4 package and includes an I²C slave interface for communication with a master processor.

Ordering Information and Content Guide appear at end of datasheet.

Key Benefits and Features

The benefits and features of ENS210, Relative Humidity and Temperature Sensor with I²C Interface are listed below:

Figure 1:
Added Value of Using ENS210

Benefits	Features
• Ultra-accurate	• Temperature sensor (±0.2°C) • Relative humidity sensor (±3.5%RH)
• Wide sensing range	• Temperature operating range (–40°C to 100°C) • Relative humidity operating range (0% to 100%)
• Wide operating voltage	• 1.71V to 3.60V
• Small foot-print	• 2.0mm x 2.0mm x 0.75mm
• Industry standard two-wire interface	• Standard (100kbit/s) and fast (400kbit/s) I²C

• Low power	• Automatic low-power standby when not measuring
	  • Active current: 6.6µA @ 1Hz (1.8V)
	  • Standby current: 40nA

• Cost effective	• Digital pre-calibrated relative humidity and temperature sensor • Output directly in %RH and Kelvin • Wide supply voltage range
• High reliability	• Long-term stability

Applications

The ENS210 applications include:

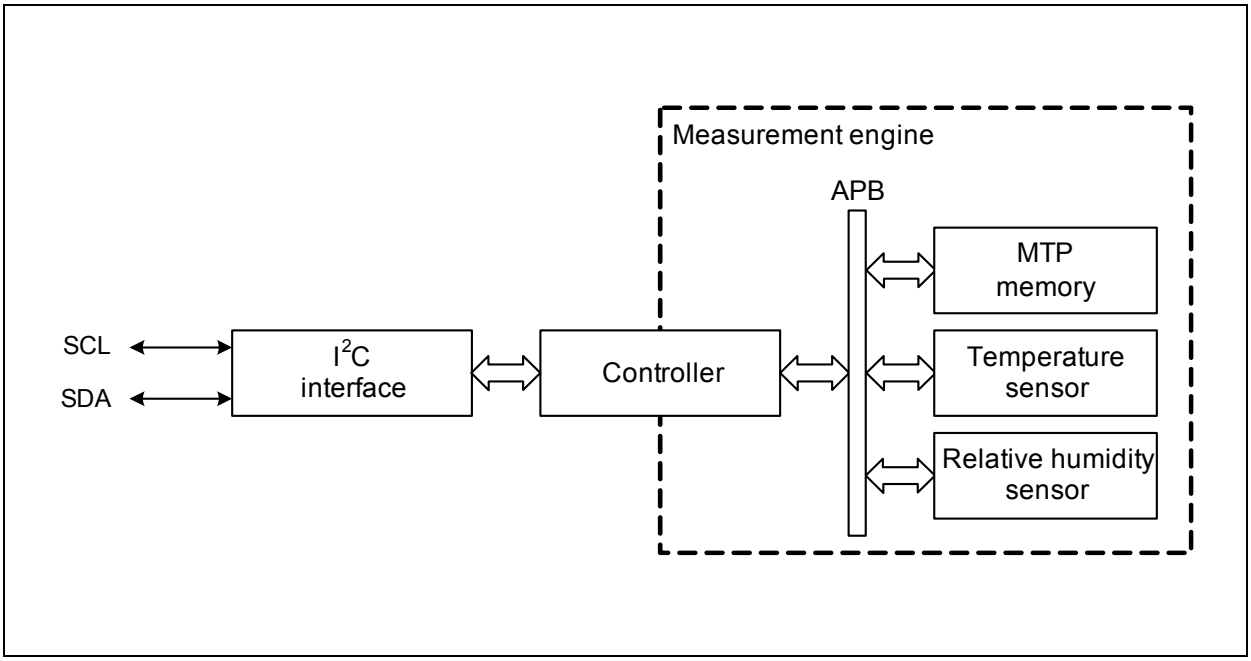
- Portable devices for personal health and wellness
- Air cleaners, air purifiers and smart thermostats
- Weather stations
- Home appliances, such as washing machines, dishwasher, and dryers
- Baby monitoring devices
- Transportation condition monitoring

Block Diagram

The internal block diagram of ENS210 is shown in Figure 2. The I^2C (communication) interface is connected to a controller which acts as the command interpreter and as bus master of the internal Advanced Peripheral Bus (APB). The memory and sensors are slaves of the APB. The MTP memory is used to store the sensor calibration parameters and unique ID.

To reduce power consumption the controller only powers the measurement engine when needed.

Figure 2:
Functional Blocks of ENS210



Pin Assignments

The ENS210 pin assignment is described in Figure 3 and Figure 4.

Figure 3:
Pin Diagram of ENS210

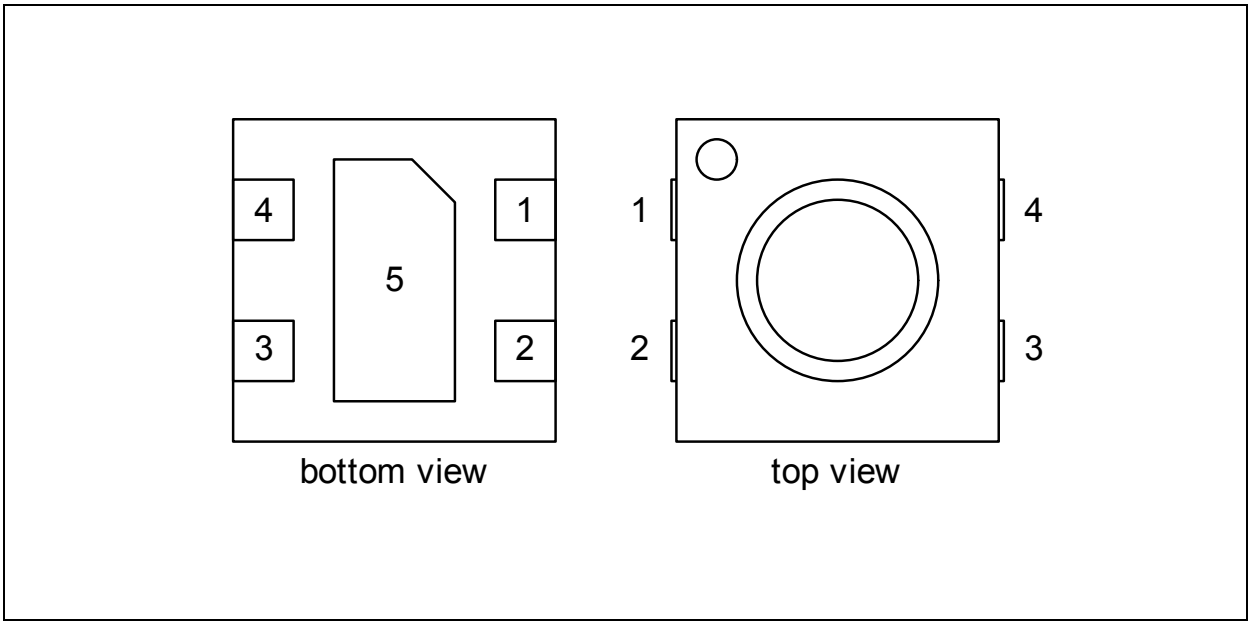


Figure 4:
Pin Description of ENS210

Pin Number	Pin Name	Description
1	V_{DD}	Supply voltage
2	SCL	I^2C bus serial clock input (SCL)
3	SDA	I^2C bus serial bidirectional data line (SDA)
4	V_{SS}	Ground supply voltage; must be connected
5	V_{SS}	Ground supply voltage; must be connected

Absolute Maximum Ratings

Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only. Functional operation of the device at these or any other conditions beyond those indicated under Electrical Characteristics is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Figure 5:
Absolute Maximum Ratings of ENS210

Symbol	Parameter	Min	Max	Units	Comments
Electrical Parameters					
V_{DD}	Supply voltage	-0.30	4.60	V	
I_{lu}	Latch-up current		100	mA	I/O; $-0.5V_{DD} < V_I < 1.5V_{DD}$; $T_j < 125°C$
Electrostatic Discharge					
ESD_{HBM}	Human body model; all pins	±2000		V	JEDEC JS-001-2014
ESD_{CDM}	Charged model device; all pins	±500		V	JEDEC JS-002-2014
Operating and Storage Conditions					
MSL	Moisture sensitivity level	1			Maximum floor life time is unlimited
T_{STRG}	Storage temperature	10	50	°C	
RH_{NC}	Relative humidity (non-condensing)	20	60	%RH	Preferably in sealed ESD bag
T_A	Operating ambient temperature	–40	100	°C	
H_A	Operating ambient relative humidity	0	100	%RH	

Electrical Characteristics

All limits are guaranteed. The parameters with min and max values are guaranteed with production tests or SQC (Statistical Quality Control) methods.

Figure 6:
Electrical Characteristics

Symbol	Parameter	Conditions	Min	Typ (1)	Max	Unit
V_{DD}	Supply voltage	Max ripple 100mV_{PP} between 0-1MHz	1.71	1.80 (3.30)	3.60	V
I_{DD}	Supply current	Standby state		0.04 (0.5)		µA
		Continuous run mode		58 (61)		µA
		T and RH measurement at 1Hz		6.6 (7.1)		µA
V_{IH}	High-level input voltage		0.7×V_{DD}		V_{DD}+0.5	V
V_{IL}	Low-level input voltage		–0.5		0.3×V_{DD}	V
I_{OL}	Low-level output current	V_{OL} = 0.4V	3			mA
		V_{OL} = 0.6V	6			mA

Note(s):

1. Values in parenthesis are for V_{DD}=3.30 V.
2. T_A = 25 °C and at 1.80 V supply voltage, unless otherwise specified

I²C Timing Characteristics

ENS210 is compliant to the I²C standard; it supports standard and fast mode as per I²C-bus specifications [UM10204, I²C-bus specification and user manual, Rev. 6, 4 April 2014].

Temperature Sensor Characteristics

Figure 7:
Temperature Sensor Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
T_{range}	Temperature range		-40		100	°C
T_{acc}	Temperature accuracy (3)	T_A = 0°C to 70°C; 3σ			0.2	°C
		T_A = –40°C to 100°C; 3σ			0.5	°C
T_{res}	Temperature resolution			0.016		°C
t_{resp}	Response time (2)	T step of 10°C by submersion (in 0°C to 70°C range); τ63 % (1)		1		s
T_{rep}	Temperature repeatability	3σ of consecutive measurement values at constant conditions	-0.1		0.1	°C
ΔT	Temperature long term drift			0.005		°C / year

Note(s):

1. 63% indicates that if a T step of 10°C, e.g. from 20°C to 30°C is made, it will take t_{resp} seconds to reach 63% of that step.
2. In an application the temperature response time depends on heat conductivity of the sensor PCB.
3. Accuracy specifications are defined before soldering of the product in an application. Refer to ENS210 application note. Maximum accuracy specification refers to 3 standard deviations assuming normal distribution of accuracy errors. After industrial calibration of sensors, each sensor is tested on typical room conditions (e.g. 25°C 45%RH) and only sensors passing the verification qualify for customer deliveries.

Relative Humidity Sensor Characteristics

Figure 8:
Relative Humidity Sensor Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
H_{range}	Relative humidity range		0		100	%RH
H_{acc}	Relative humidity accuracy(3)	T_A = 25°C; RH = 20%RH to 80%RH; excluding hysteresis		2.2	3.5	%RH
		T_A = 25°C; RH = 0%RH to 100%RH; excluding hysteresis		4	5	%RH
H_{res}	Relative humidity resolution			0.03		%RH
t_{resp}	Response time(4)	RH step of 20%RH (in 40%RH to 80%RH range); τ63%(1); 1m/s flow; T_A = 25°C		3	5	s
H_{hys}	Relative humidity hysteresis	T_A = 25°C; RH = 20%RH to 90%RH; 30minutes exposure time		±0.7		%RH
H_{rep}	Relative humidity repeatability	3σ of consecutive measurement values at T_A = 25°C and RH = 40%RH		±0.1		%RH
ΔH	Relative humidity long term drift(2)	T_A = 25°C		0.25		%RH / year

Note(s):

1. 63% indicates that if an RH step of 20%RH is made, e.g. from 40%RH to 60%RH, it will take t_{resp} seconds to reach 63% of that step.
2. Values are linearized averages over the lifetime of the product. Due to non-linear behavior a larger drift is expected in the first years.
3. Typical and maximum accuracy specification refers to, respectively, 2 and 3 standard deviations, assuming normal distribution of accuracy error.
4. Device only performance. Application response time will depend on the design-in of the sensor

System Timing Characteristics

Figure 9:
System Timing Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$t_{booting}$	Booting time (1)			1	1.2	ms
t_{conv}	Conversion time	T only, single shot (includes $t_{booting}$)		105	110	ms
		T only, continuous		104	109	ms
		T and RH, single shot (includes $t_{booting}$)		122	130	ms
		T and RH, continuous		225	238	ms

Note(s):

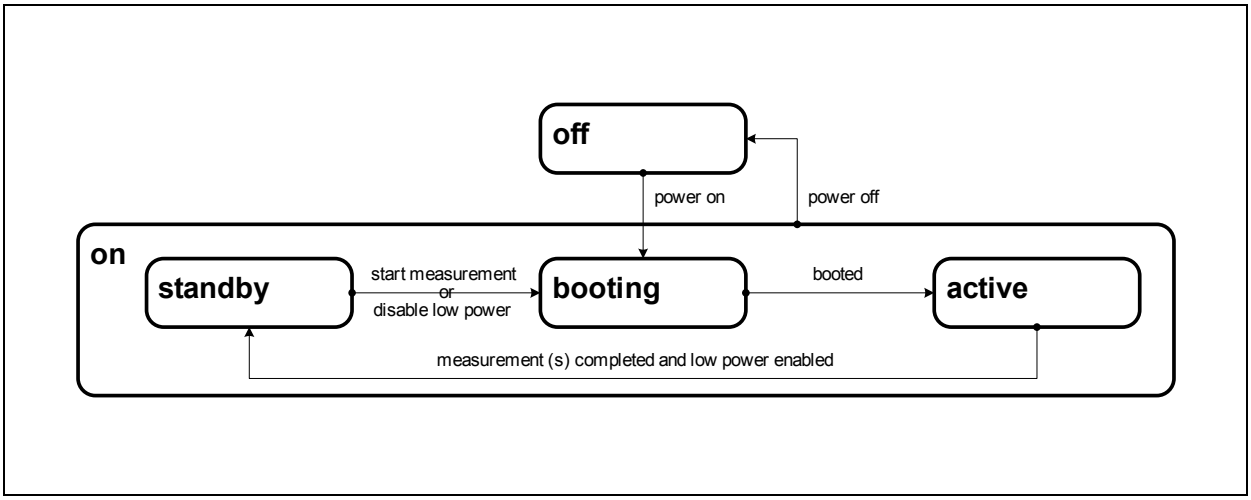
1. Time in transient state *booting* (see Figure 10).

Functional Description

The ENS210 integrates two sensor blocks: temperature and relative humidity.

The device is normally in the *standby* state (Figure 10): the measurement engine (see Figure 2) is unpowered, but the I²C interface is operational and register write/read operation can be performed. When a measurement command is given, the device is first *booting* to *active* then it starts a measurement. When the measurement is completed, the device returns to the *standby* state. Since the I²C interface is operational in *standby*, the measurement result can be read out.

Figure 10:
The ENS210 Power States



In continuous run mode (see Register SENS_RUN) or when low power is disabled (see Register SYS_CTRL), the device remains in *active* state.

The system power status is observable (see Register SENS_STAT).

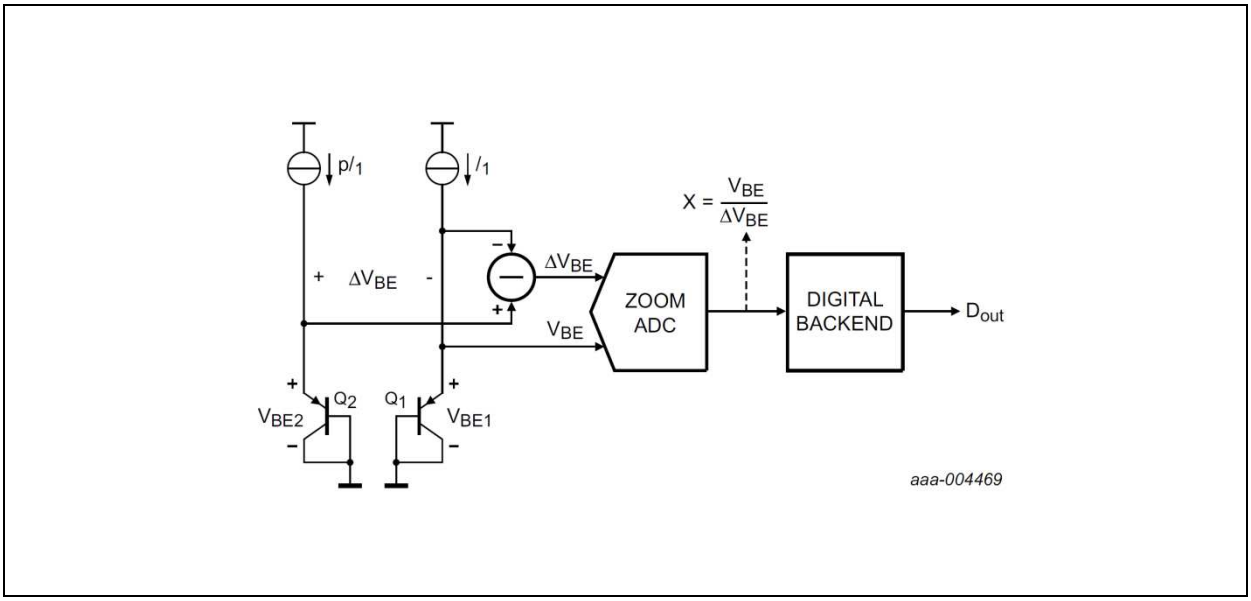
When powering up from *off*, the device is first *booting* to *active*, but then falls immediately back to *standby* (since no measurement is pending, and by default low power is enabled).

Note that the *booting* state is a transient state (the system automatically transitions to the next state – *active*); the booting time is given in Figure 9.

Temperature Sensor

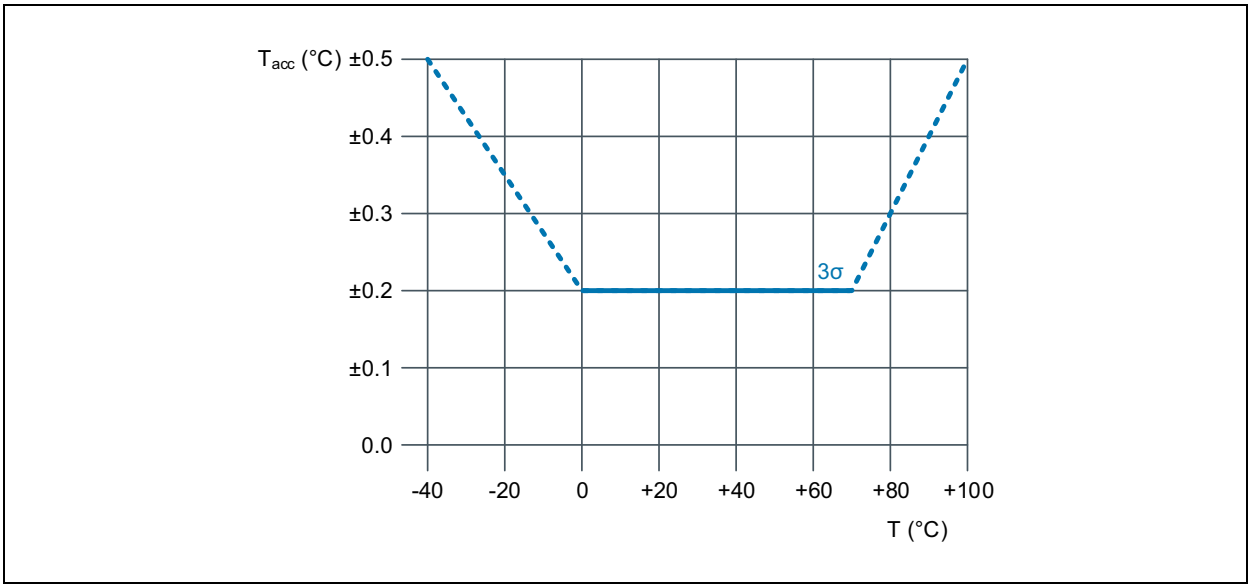
The temperature sensor block (Figure 11) determines the ambient temperature, and outputs a calibrated value in Kelvin.

Figure 11:
Band Gap Temperature Measurement



The temperature is measured using a high-precision (12 bits) zoom-ADC. The analog part is able to measure a strongly temperature dependent $X = V_{BE}/\Delta V_{BE}$. The X is found by first applying a coarse search (successive approximation), and then a sigma-delta in a limited range. The accuracy of the sensor is shown in Figure 12. The conversion time is shown in Figure 9.

Figure 12:
Absolute Accuracy of the Temperature Sensor



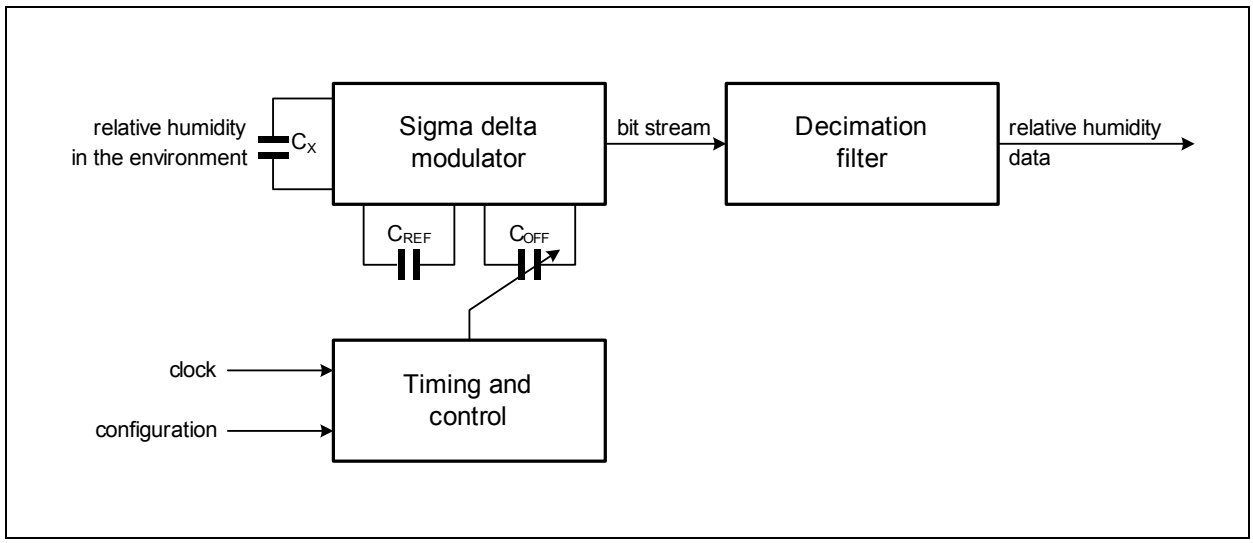
Note(s):

1. Dash line indicates natural physical behavior

Relative Humidity Sensor

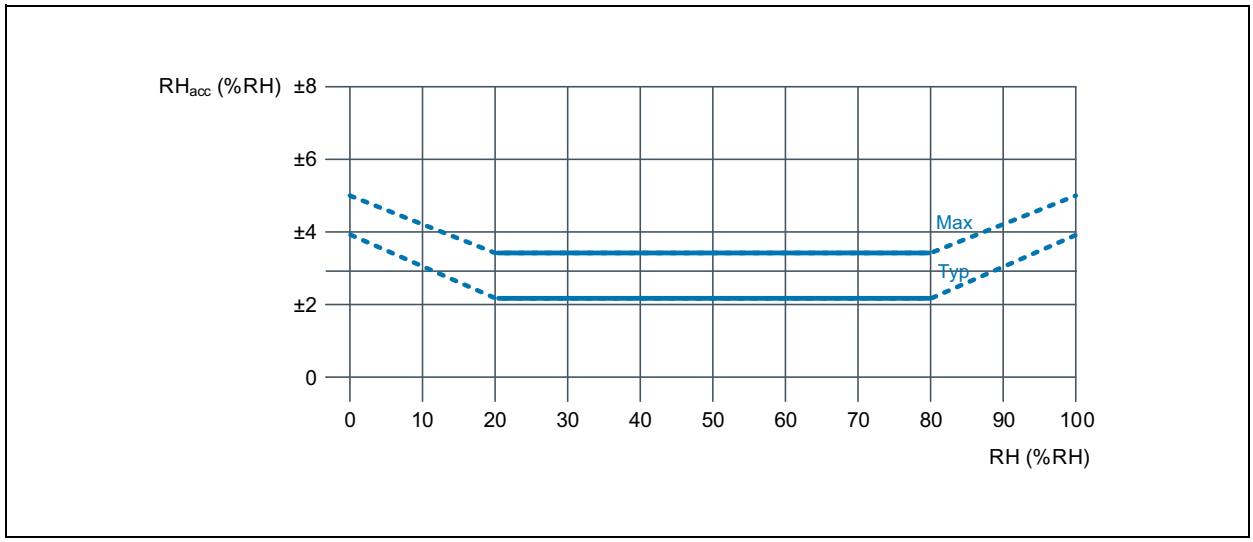
The relative humidity sensor as shown in Figure 13 determines the ambient relative humidity and outputs a calibrated value in %RH. The transducer (the C_X on the top left) consists of a large-area capacitor covered with a humidity-sensitive material. The capacitance change is proportional to the change in relative humidity, and has a linear dependence on temperature. The capacitance is measured by a high-precision 2^{nd} order sigma-delta converter.

Figure 13:
Relative Humidity Sensor



Reading the relative humidity sensor will output a temperature compensated value. The accuracy of the sensor is shown in Figure 14. The conversion time is shown in Figure 9.

Figure 14:
Absolute Accuracy of the Relative Humidity Sensor at 25°C



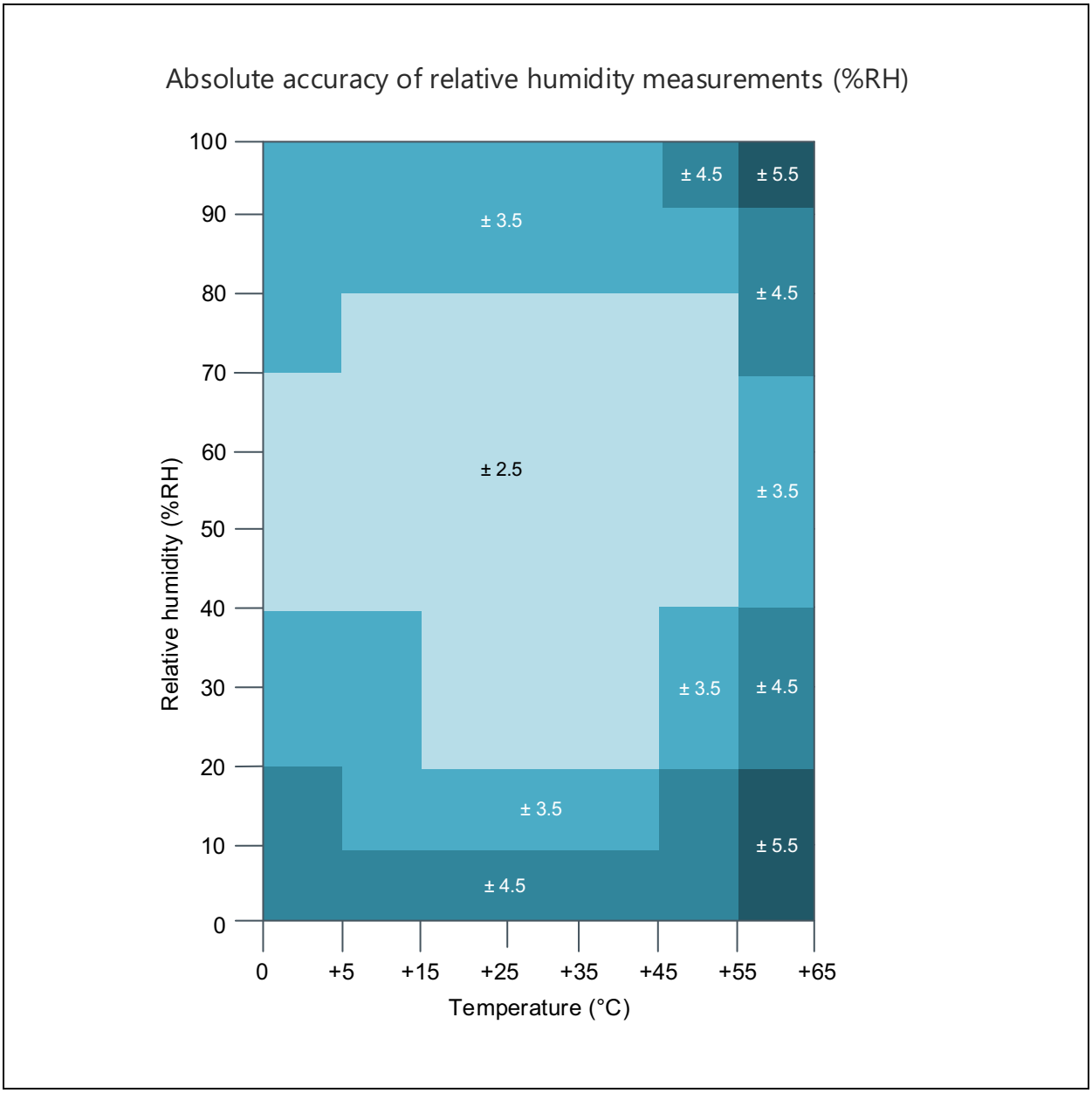
Note(s):

1. Dash line indicates natural physical behavior

RH Accuracy at Various Temperatures

Typical RH accuracy at 25°C is defined in Figures 8 and 14. The relative humidity accuracy has also been evaluated at temperatures other than 25°C. The values shown in Figure 15 are an indication only, which may be important for your application, but are not guaranteed.

Figure 15:
Accuracy of Relative Humidity Measurements (%RH) as Function of Temperature and Relative Humidity



The I²C Interface

The ENS210 is an I²C slave device. The I²C interface supports standard (100kbit/s) and fast (400kbit/s) mode.

Details on I²C protocol is according to *I²C-bus specifications [UM10204, I²C-bus specification and user manual, Rev. 6, 4 April 2014]*.

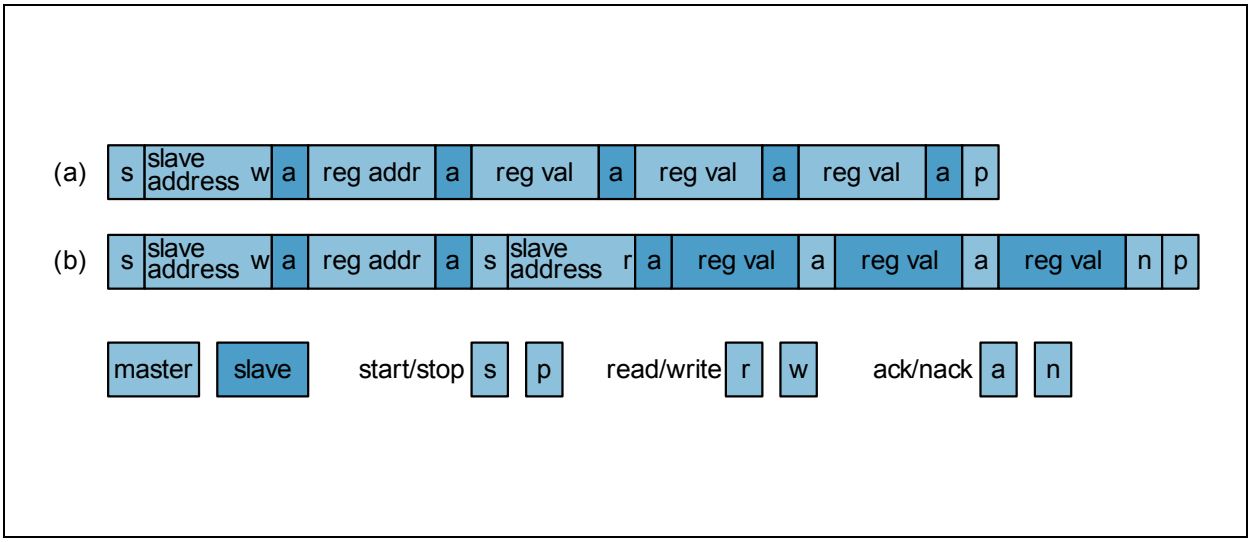
The device applies all mandatory I²C protocol features for slaves: START, STOP, Acknowledge, 7-bit slave address. ENS210 does not use clock stretching.

None of the other optional features (10-bit slave address, General Call, Software reset, or Device ID) are supported, nor are the master features (Synchronization, Arbitration, START byte).

I²C Operations on Registers

The ENS210 uses a register model to interact with it. This means that an I²C master can write a value to one of the registers of a slave, or that it can read from one of the registers of the slave. In the ENS210, registers are addressed using 1 byte. The values stored in a register are also 1 byte. However, the ENS210 implements "auto increment" which means that it is possible to read, for example, two bytes by supplying the address of the first byte and then reading two bytes.

Figure 16:
I²C Transaction Formats



A typical *write* transaction (see Figure 16 a) therefore has the following format. The master initiates a transaction with a so-called start condition "s". This blocks the bus. Next, the master sends the 7 bits ENS210 *slave address* followed by a 1 bit direction (a 0 indicating write "w"). This byte is acknowledged "a" by the slave. The master continues by sending the 8 bit *register address*, which is acknowledged by the slave.

This register address is stored in an internal CRA register ("Current Register Address"). Finally, the master sends the 8 bit *register value*, which is acknowledged by the slave (or nack'ed when the address is not writeable). This value is written to the register pointed to by the CRA, and the CRA is incremented by 1. Optionally, the master sends more 8 bit values, for the next registers (auto incrementing CRA), each of which is (n)ack'ed by the slave. Finally, the master generates a stop condition "p", unblocking the bus for other transactions.

A *read* transaction (see Figure 16 b) starts with a write (of the register address), followed by a read. Consequently, it has the following format. The master initiates the transaction with a start condition. Next, the master sends the 7 bits ENS210 *slave address* followed by a 1 bit direction (a 0 indicating write). This byte is acknowledged by the slave. The master continues by sending the 8 bit *register address*, which is acknowledged by the slave and stored in the CRA register. Then the master sends another start condition (a so-called repeated start condition, keeping the bus blocked) followed by the 7 bits ENS210 *slave address* followed by a 1 bit direction (a 1 indicating read "r"), which is acknowledged by the slave. Next, the slave sends an 8 bits *register value* from the register pointed to by the CRA register, and the CRA is incremented by 1. This byte is acknowledged by the master. The master may read another 8 bits (auto increment feature) from the slave and acknowledge that, until the master sends a nack "n" followed by a stop to unblock the bus.

The ENS210 has an 8 bit address space, potentially addressing 256 registers. In reality, only few addresses are actually backed by a register (see Register Overview). All other addresses are *reserved*. A write transaction to a reserved (or read-only) register causes a not-acknowledge. A read transaction for a reserved register will return a 0.

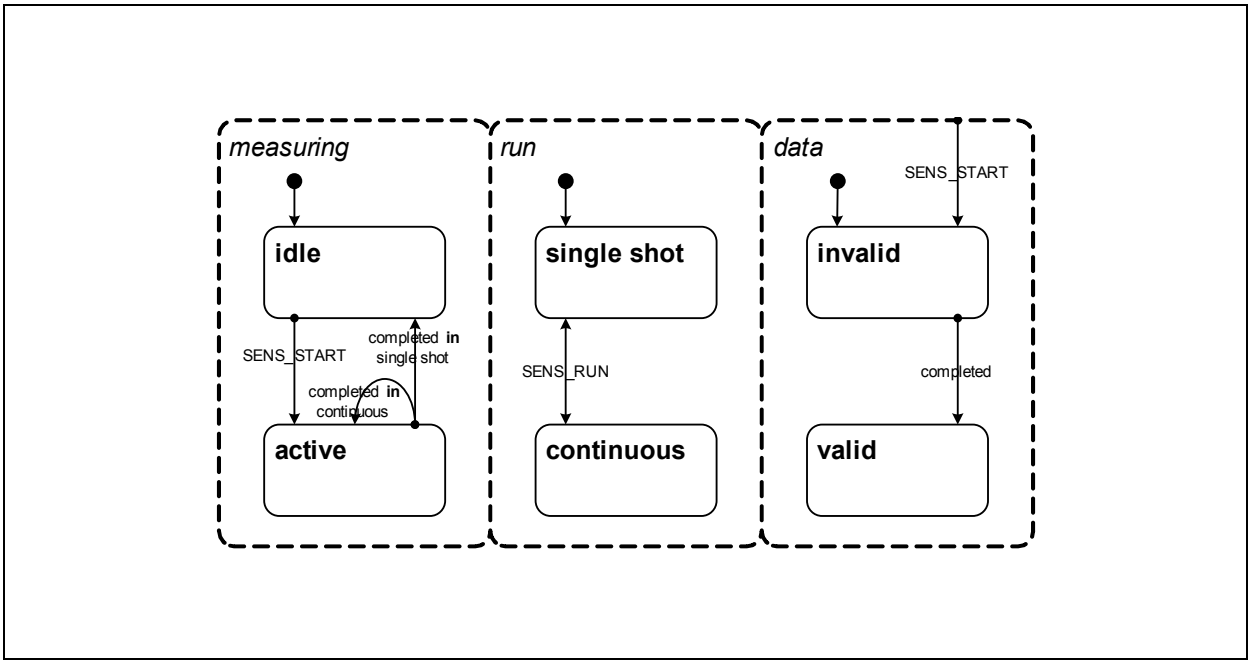
The I²C Slave Address

The ENS210 is an I²C slave device with a fixed slave address of 0x43. This means that the first byte after a start condition is 1000 011x, where x indicates the data direction, so 0x86 (1000 0110) for write and 0x87 (1000 0111) for read.

Sensor Control

The ENS210 contains a temperature and a relative humidity sensor. Both sensors have two run modes: single shot run mode and continuous run mode (enabled via SENS_RUN), see Figure 17.

Figure 17:
The Sensor Modes



When in the *single shot* run mode, starting a measurement is under control of the master. By default a sensor is idle; it can be started by writing a 1 to the corresponding bit in SENS_START. After a start, the sensor stops when the measurement is completed. Whether a sensor is idle or active measuring can be detected by reading SENS_STAT. The measured values can be obtained via their respective readout registers (T_VAL and H_VAL). Writing to SENS_STOP in single shot has no effect.

When in the *continuous* run mode, the sensor performs measurement after measurement after a 1 is written to the corresponding bit in SENS_START. The result of each measurement is stored in the aforementioned readout registers. Writing 1 to the corresponding bit in SENS_STOP stops the repeat cycle after the ongoing measurement is completed.

The device operates in a step-wise way. In each step, either one or both sensors are active. The step ends when the measurement(s) are completed. For the next step, the device

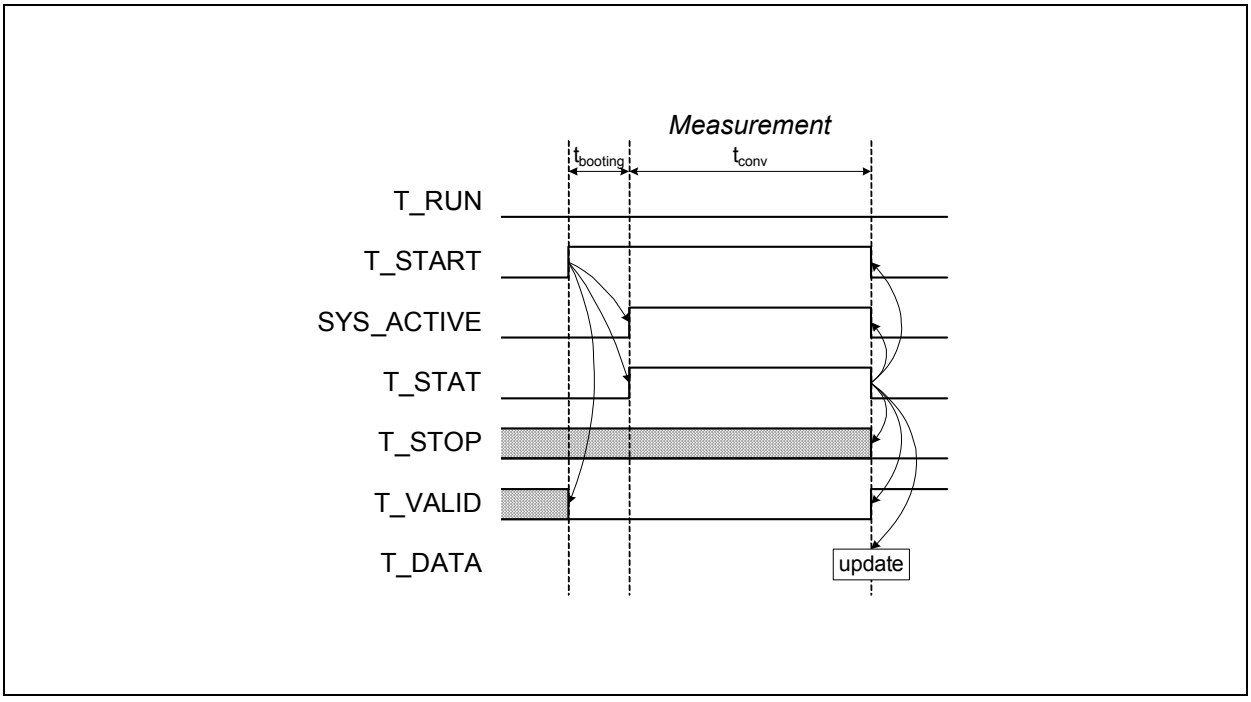
inspects its register settings, and either one or both sensors are activated again, or there is no measurement request and the device goes into standby (unless low power is disabled by SYS_CTRL).

This means that multiple writes to START during a step have no effect; the measurement is started once, and only a write to START after the measurement has completed starts the measurement again. Similarly, multiple writes to STOP have no effect; when the measurement completes (in continuous mode) the stop request is effectuated once. When START and STOP are both requested, the measurement is started, and when completed, stopped.

Sensor Timing

There are differences between single shot measurements and continuous measurements. Figure 18 shows the timing of a single shot T measurement.

Figure 18:
Single Shot Temperature Measurement



Signal T_RUN is written low to select a single shot measurement. Note that T_STOP is typically low (cleared by a previous measurement), but its state is ignored in a single shot measurement. T_START is written high to start measuring: T_VALID in T_VAL is cleared and the device starts *booting* to *active*. Once active SYS_ACTIVE goes high, and measurement starts (T_STAT goes high).

When the measurement is completed (T_STAT goes low) the data register (T_DATA) becomes valid (T_VALID goes high) and the device goes back to *standby* (SYS_ACTIVE goes low). The T_START and T_STOP are cleared.

Figure 19:
Continuous Temperature Measurement

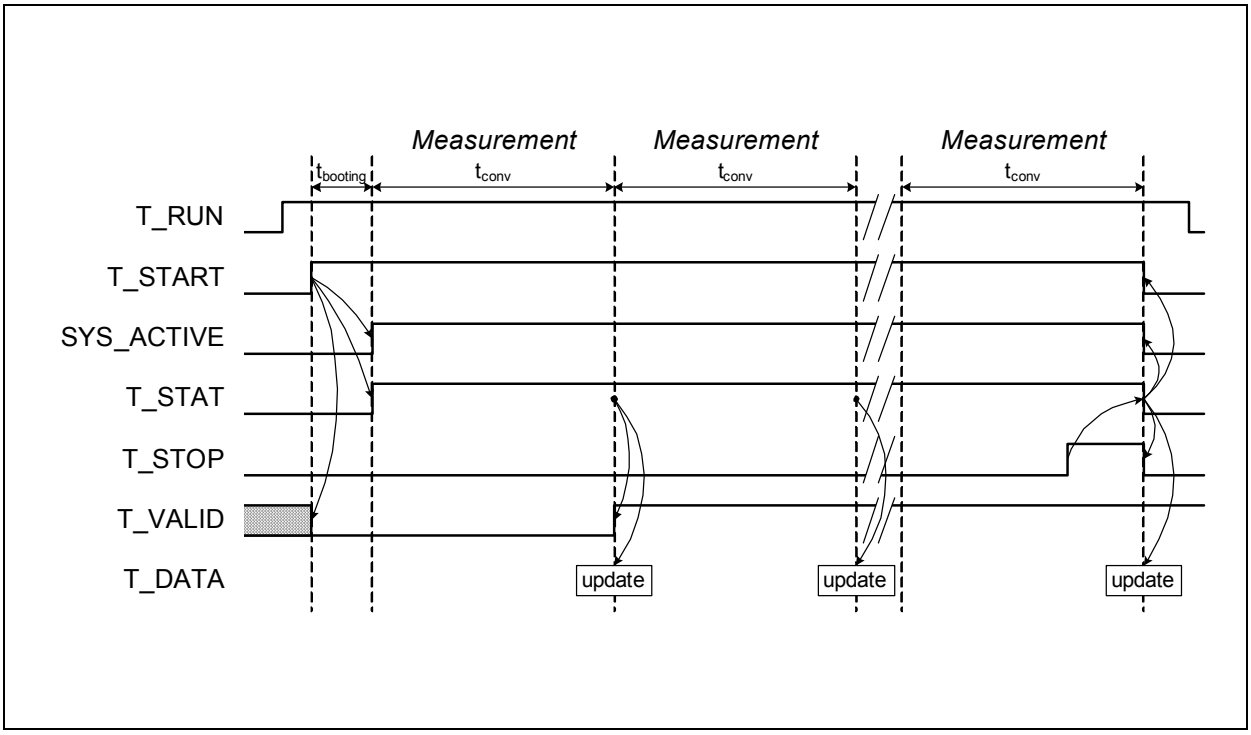


Figure 19 shows the timing of a continuous T measurement.

Signal T_RUN is written high to select a continuous measurement. Note that T_STOP is typically low (cleared by a previous measurement), and it should stay low otherwise continuous mode will stop after one measurement. T_START is written high to start measuring: T_VALID in T_VAL is cleared and the device starts *booting* to *active*. Once *active* SYS_ACTIVE goes high, and measurement starts (T_STAT goes high).

When the first measurement is completed the data register (T_DATA) becomes valid (T_VALID goes high), and the device starts a new measurement. When the next measurement is completed the data register (T_DATA) is updated; T_VALID stays high. The device starts a new measurement.

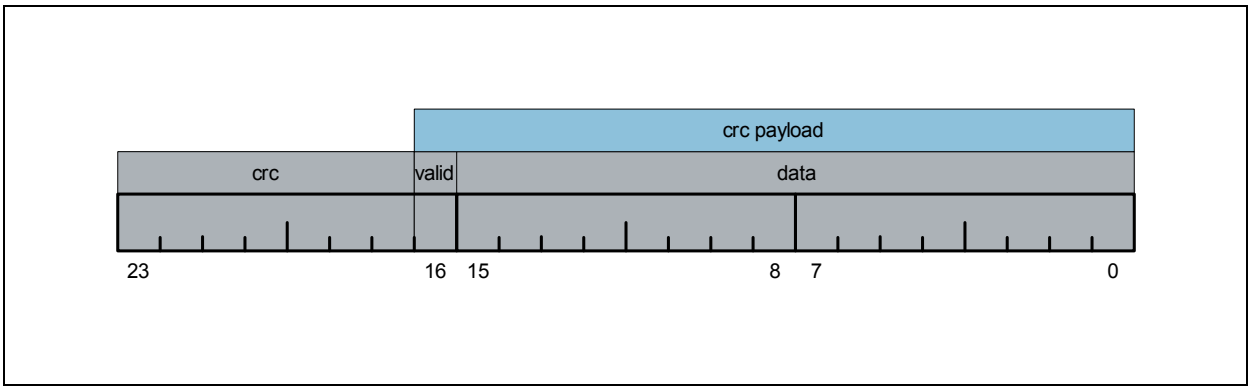
At some point in time, a stop command is given (T_STOP is written high). As soon as the current measurement is completed, the data register (T_DATA) is once more updated and the device goes back to *standby* (SYS_ACTIVE goes low). The T_START and T_STOP are cleared.

Note that writes to the SENS_XXX registers only take effect when no measurement is ongoing. In other words, measurements are always sequential (so we can have three types: T only, RH only or T and RH and changes occur when the measurements are finished.

The Sensor Readout Registers

The sensor readout registers (T_VAL and H_VAL) consist of three parts: the actual measured data, a valid flag and a checksum (see Figure 20). It is not mandatory to read the valid flag or the checksum when reading the data.

Figure 20:
The Layout of the Sensor Readout Registers



The *checksum* is a cyclic redundancy check over the data and the valid flag; the stored checksum is the result of CRC-7 (polynomial x7+x3+1, see *https://en.wikipedia.org/wiki/Cyclic_redundancy_check) with 0x7F as initial vector (i.e. with all bits flipped), see* Computing CRC-7 *for sample C code.*

The *valid* flag is cleared when a measurement is started (irrespective of the run mode). Once the measurement is completed the valid flag is set. In continuous mode, a new measurement is then started without clearing the valid flag; so data is always valid after the first measurement (but it might be several milliseconds old).

The *data* field is a 16 bits fixed point number, whose format and unit depends on the sensor (see Register T_VAL and Register H_VAL).

To ensure consistent view, these multi-byte readout registers are double buffered. When the first byte (i.e. the byte with the lowest register address) is read, the device copies all bytes from the measurement registers to the I^2C registers, and then the value from the first I^2C register is returned. Reads to the other bytes of the multi-byte register (i.e. with higher register addresses) are always directly from the I^2C registers.

Computing CRC-7

CRC algorithm uses a 7 bit polynomial (see lines 4, 5, and 6), and a 17 bit payload. The `crc7()` function below uses the following constants defining the CRC width, (the coefficients of the) polynomial and the initial vector (start value of the CRC), and some constants describing the payload data size.

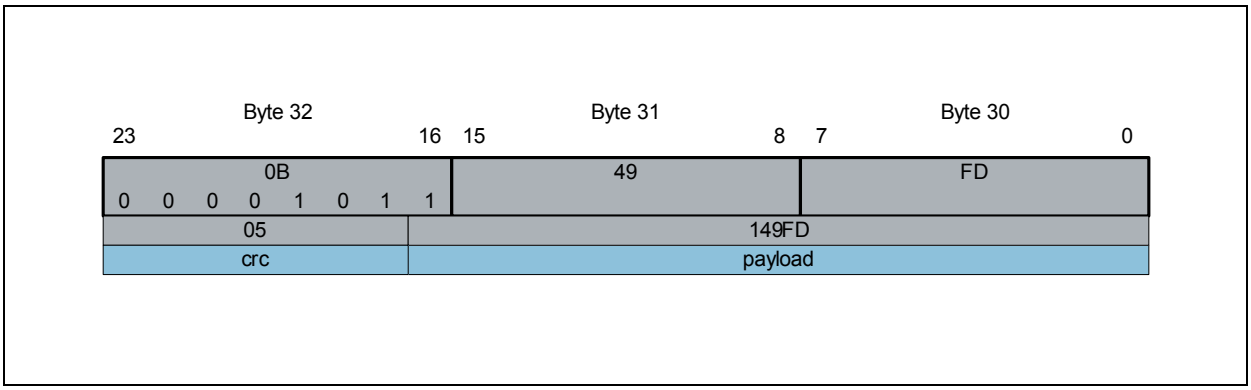
```
//                    7654  3210
// Polynomial    0b   1000  1001 ~ x^7+x^3+x^0
//               0x      8     9
#define CRC7WIDTH   7       // 7 bits CRC has polynomial of 7th order (has 8 terms)
#define CRC7POLY    0x89    // The 8 coefficients of the polynomial
#define CRC7IVEC    0x7F    // Initial vector has all 7 bits high
// Payload data
#define DATA7WIDTH  17
#define DATA7MASK   ((1UL<<DATA7WIDTH)-1) // 0b 0 1111 1111 1111 1111
#define DATA7MSB    (1UL<<(DATA7WIDTH-1)) // 0b 1 0000 0000 0000 0000
```

The `crc7(val)` function returns the CRC-7 of a 17 bits value *val*.

```
// Compute the CRC-7 of 'val' (should only have 17 bits)
uint32_t crc7( uint32_t val ) {
    // Setup polynomial
    uint32_t pol= CRC7POLY;
    // Align polynomial with data
    pol = pol << (DATA7WIDTH-CRC7WIDTH-1);
    // Loop variable (indicates which bit to test, start with highest)
    uint32_t bit = DATA7MSB;
    // Make room for CRC value
    val = val << CRC7WIDTH;
    bit = bit << CRC7WIDTH;
    pol = pol << CRC7WIDTH;
    // Insert initial vector
    val |= CRC7IVEC;
    // Apply division until all bits done
    while( bit & (DATA7MASK<<CRC7WIDTH) ) {
        if( bit & val ) val ^= pol;
        bit >>= 1;
        pol >>= 1;
    }
     return val;
}
```

Suppose that T_VAL (address 30, 31 and 32) reads FD 49 0B, corresponding (little endian) with the number 0B49FD, see Figure 21. This leads to a CRC of 05 over a payload of 149FD. See the next paragraph for details on processing this data.

Figure 21:
T_VAL Readout



Processing T_VAL and H_VAL

This paragraph shows a possible implementation of reading T and RH.

The following fragment starts a combined single shot measurement, waits and reads the measurement results. It assumes the availability of `i2c_reg_write` and `i2c_reg_read` primitives as well as a sleep routine (`rtk_tsk_sleep`). The format specifiers in the `printf`'s are kept simple (`%d` instead of `%ld` or even `PRId32` from `inttypes.h`); they need adaptation on e.g. 16 bits platforms.

```
// Record I²C transaction status
bool i2c_ok= true;
// Start T and H (write 03 to register 22 in device 86)
uint8_t wbuf[]= { 0x03 };
i2c_ok &= i2c_reg_write(0x86, 0x22, wbuf, sizeof wbuf );
// Wait for measurements to complete
#define   CONVERSION_TIME_T_H_MS   130
rtk_tsk_sleep(CONVERSION_TIME_T_H_MS);
// Read T and H (read 6 bytes starting from 0x30 in device 86)
uint8_t rbuf[6];
i2c_ok &= i2c_reg_read(0x86, 0x30, rbuf, sizeof rbuf );
// Extract T_VAL and H_VAL (little endian), assumes 32 bits wordsize
uint32_t t_val= (rbuf[2]<<16) + (rbuf[1]<<8) + (rbuf[0]<<0);
uint32_t h_val= (rbuf[5]<<16) + (rbuf[4]<<8) + (rbuf[3]<<0);
```

The following fragment processes the T measurement as available in `t_val`. It relies on the `crc7()` function as shown previously.

```
// Extract (and print) the fields
uint32_t t_data = (t_val>>0 ) & 0xffff;
uint32_t t_valid= (t_val>>16) & 0x1;
uint32_t t_crc  = (t_val>>17) & 0x7f;
printf("ENS210: T: %06x %02x %01x %04x\n", t_val, t_crc, t_valid, t_data );
// Check the CRC
uint32_t t_payl = (t_val>>0 ) & 0x1ffff;
bool t_crc_ok= crc7(t_payl)==t_crc;
// Convert to float (and print)
float TinK = (float)t_data / 64; // Temperature in Kelvin
float TinC = TinK - 273.15;      // Temperature in Celsius
float TinF = TinC * 1.8 + 32.0;  // Temperature in Fahrenheit
printf("ENS210: T: (i2c=%d crc=%d valid=%d) %5.1fK %4.1fC %4.1fF\n", i2c_ok, t_crc_ok, t_valid, TinK, TinC, TinF );
```

The following fragment processes the RH measurement as available in `h_val`. It is similar to the `t_val` processing.

```
// Extract (and print) the fields
uint32_t h_data = (h_val>>0 ) & 0xffff;
uint32_t h_valid= (h_val>>16) & 0x1;
uint32_t h_crc  = (h_val>>17) & 0x7f;
printf("ENS210: H: %06x %02x %01x %04x\n", h_val, h_crc, h_valid, h_data );
// Check the CRC
uint32_t h_payl = (h_val>>0 ) & 0x1ffff;
bool h_crc_ok= crc7(h_payl)==h_crc;
// Convert to float (and print)
float H = (float)h_data/512;     // relative humidity (in %)
printf("ENS210: H: (i2c=%d crc=%d valid=%d) %2.0f%%\n", i2c_ok, h_crc_ok, h_valid, H );
```

If registers 30 to 35 would contain `fd 49 0b 6c 2e f5` (i.e. `T_VAL` in blue and `H_VAL` in green) the code would print

```
ENS210: T: 0b49fd 05 1 49fd
ENS210: T: (i2c=1 crc=1 valid=1) 296.0K 22.8C 73.0F
ENS210: H: f52e6c 7a 1 2e6c
ENS210: H: (i2c=1 crc=1 valid=1) 23%
```

Reading PART_ID and UID

The first 2 registers (PART_ID and UID) are only available in *active* state. There are two ways to read them:

- Dedicated read action
  - Disable low power (set LOW_POWER to 0)
  - Wait for $t_{booting}$ to get into *active* state (check SYS_ACTIVE to be 1)
  - Read the ID register(s)
  - Re-enable low power (set LOW_POWER to 1)
- Piggybacking on a measurement
  - Start a measurement (write 0b01, 0b10, or 011 to SENS_START)
  - Wait for $t_{booting}$ to get into *active* state (check SYS_ACTIVE to be 1)
  - Read the ID register(s)
  - Ensure the device is still in *active* state (check SYS_ACTIVE to be 1)

Register Description

This section describes the I²C registers of the ENS210.

Register Overview

Note that some registers are actually spread over multiple addresses. For example, T_VAL at address 30 is spread over 3 addresses (its "Size" is 3). This could be rephrased as follows: there are three registers T_VAL0, T_VAL1, and T_VAL2 at addresses 30, 31, and 32 respectively.

Figure 22:
Register Overview

Address	Name	Size	Access	Description
0x00	PART_ID	2	Read (active only)	Identifies the part as ENS210
0x02	<unused>	2	Read	
0x04	UID	8	Read (active only)	Unique identifier
0x0C	<reserved>	4		
0x10	SYS_CTRL	1	Read/Write	System configuration
0x11	SYS_STAT	1	Read	System status
0x12	<reserved>	14		
0x21	SENS_RUN	1	Read/Write	The run mode (single shot or continuous)
0x22	SENS_START	1	Write	Start measurement
0x23	SENS_STOP	1	Write	Stop continuous measurement
0x24	SENS_STAT	1	Read	Sensor status (idle or measuring)
0x25	<reserved>	11		
0x30	T_VAL	3	Read	Temperature readout
0x33	H_VAL	3	Read	Relative humidity readout
0x36	<reserved>	202		

Detailed Register Description

Register PART_ID (Address 0x00)

This 2 byte register identifies the part number in little endian (ENS210). This register is only available in *active* state; see Reading PART_ID and UID for instructions of reading it.

Figure 23:
Register PART_ID

Address 0x00		PART_ID		
Bits	**Field Name**	**Default**	**Access**	**Field Description**
15:0	PART_ID	0x0210	Read	Identifies this device as an ENS210

Register UID (Address 0x04)

This 8 byte register uniquely identifies a single device among all ENS210 devices. This register is only available in *active* state; see Reading PART_ID and UID for instructions of reading it.

Figure 24:
Register UID

Address 0x04		UID		
Bits	**Field Name**	**Default**	**Access**	**Field Description**
63:0	UID	Varies	Read	Unique device id

Register SYS_CTRL (Address 0x10)

This 1 byte register controls the system.

Figure 25:
Register SYS_CTRL

Address 0x10		SYS_CTRL		
Bits	**Field Name**	**Default**	**Access**	**Field Description**
7	RESET	0	Write	Write 1 to reset the device
6:1	<reserved>	0b000000	Read/Write	Keep to 0's
0	LOW_POWER	0b1	Read/Write	Controls the automatic low power. 0: Disabled (device stays in *active*) 1: Enabled (device goes to *standby* when measurement complete)

Register SYS_STAT (Address 0x11)

This 1 byte register indicates the system status.

Figure 26:
Register SYS_STAT

Address 0x11		SYS_STAT		
Bits	**Field Name**	**Default**	**Access**	**Field Description**
7:1	<reserved>	0b0000000	Read	Reads 0's
0	SYS_ACTIVE	0b1	Read	The system power state 0: System is in *standby* or *booting* state 1: System is in *active* state

Register SENS_RUN (Address 0x21)

This 1 byte register configures the run modes (single shot or continuous) of the sensors.

Figure 27:
Register SENS_RUN

Address 0x21		SENS_RUN		
Bits	**Field Name**	**Default**	**Access**	**Field Description**
7:2	<reserved>	0b000000	Read/Write	Keep to 0's
1	H_RUN	0b0	Read/Write	The run mode of the relative humidity sensor 0: Relative humidity sensor operates in single shot mode 1: Relative humidity sensor operates in continuous mode
0	T_RUN	0b0	Read/Write	The run mode of the temperature sensor 0: Temperature sensor operates in single shot mode 1: Temperature sensor operates in continuous mode