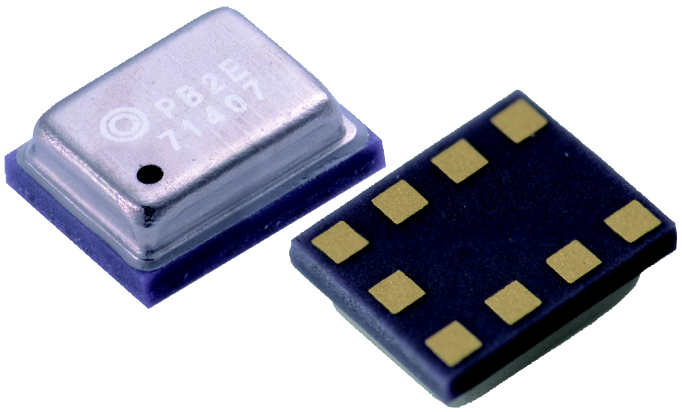
OMRON

Digital Barometric Pressure Sensor

2SMPB-02E

User's Manual

Digital Barometric Pressure Sensor



A277-E1-01

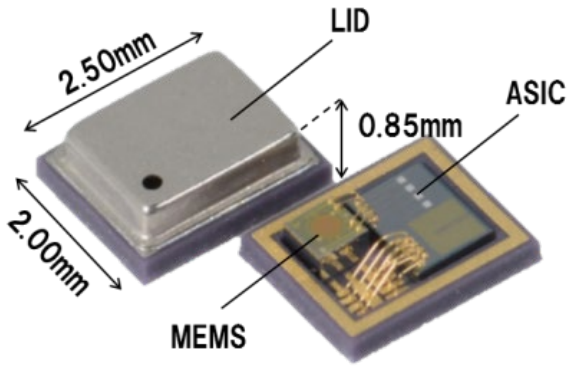
Contents

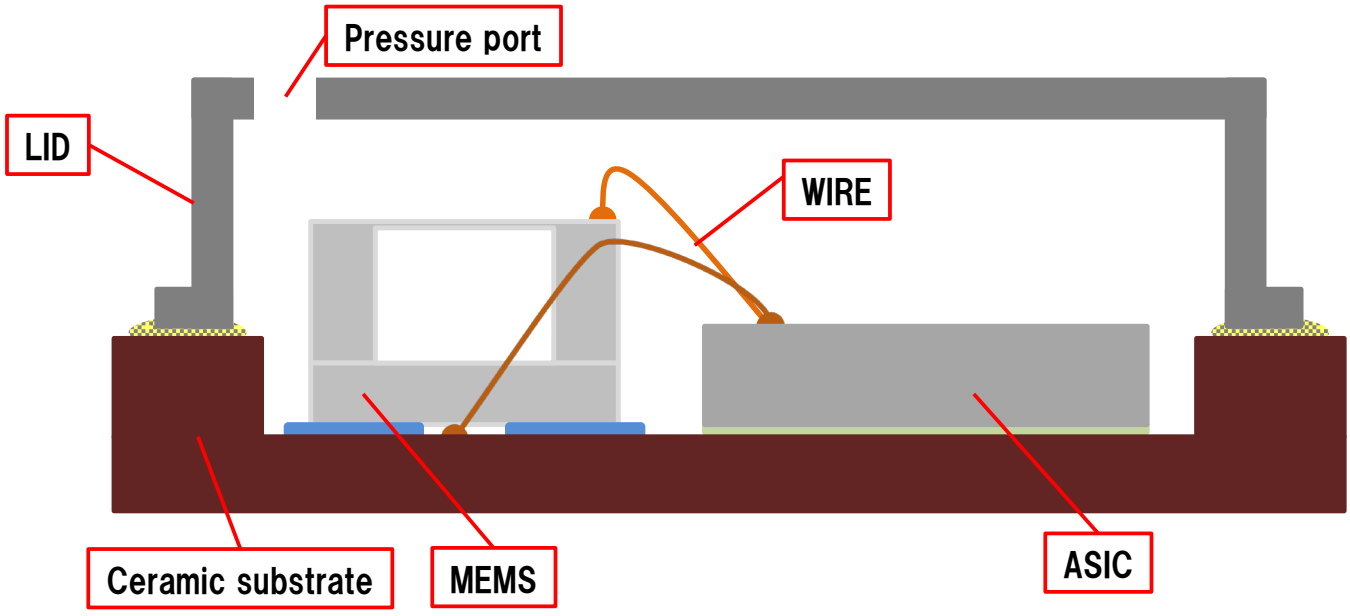
1. Outline

This user's manual is intended to demonstrate how to use and interface with Omron's digital barometric pressure sensor(2SMPB-02E). It should be noted that this document is intended to supplement the datasheet, which should be referenced when using the sensor.

2. Structure

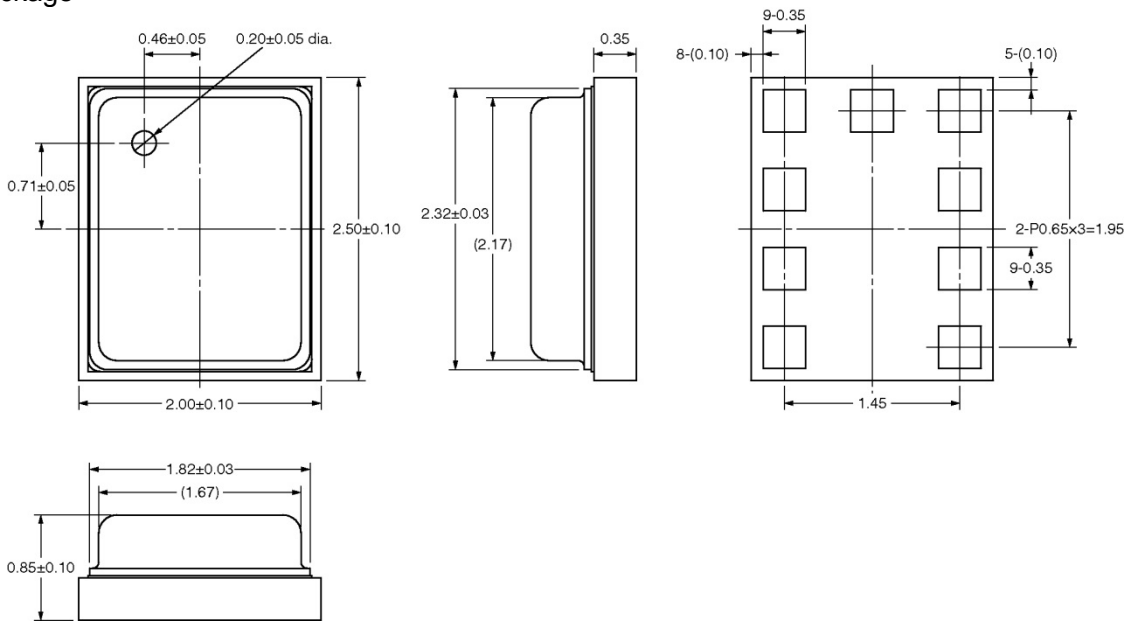
Fig.1 shows the interior of the package of the digital barometric pressure sensor (2SMPB-02E). And Fig.2 shows the internal cross-section view of it. The MEMS sense the atmospheric pressure through the pressure port.



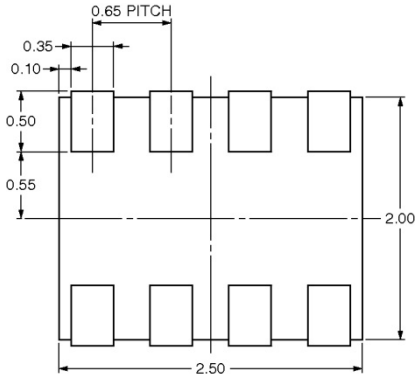


3. Dimensions

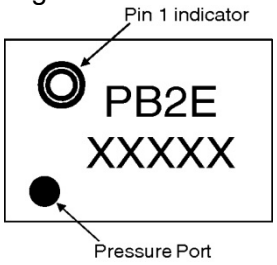
3.1. Package



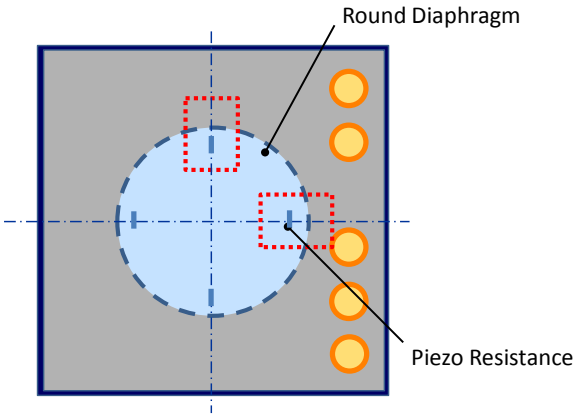
3.2. Mounting PAD Dimensions
(Top View) : Recommended



3.3. Marking Structure

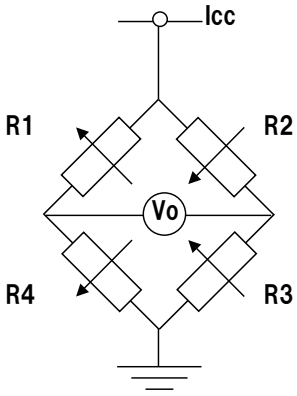


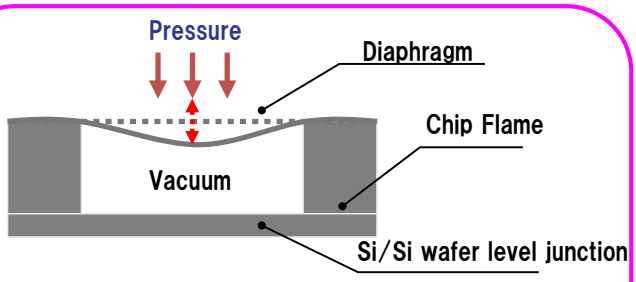
4. Principle of Pressure Detection



Schematic Top Chip

Detection Theory : Full-bridge circuit





Schematic Cross-sectional Chip

Bridge output voltage

$$Vo = \left(\frac{R_4}{R_1 + R_4} - \frac{R_3}{R_2 + R_3} \right) \cdot Vi$$

When tensile stress is applied to the piezo-resistance

Ri → Ri + ΔR

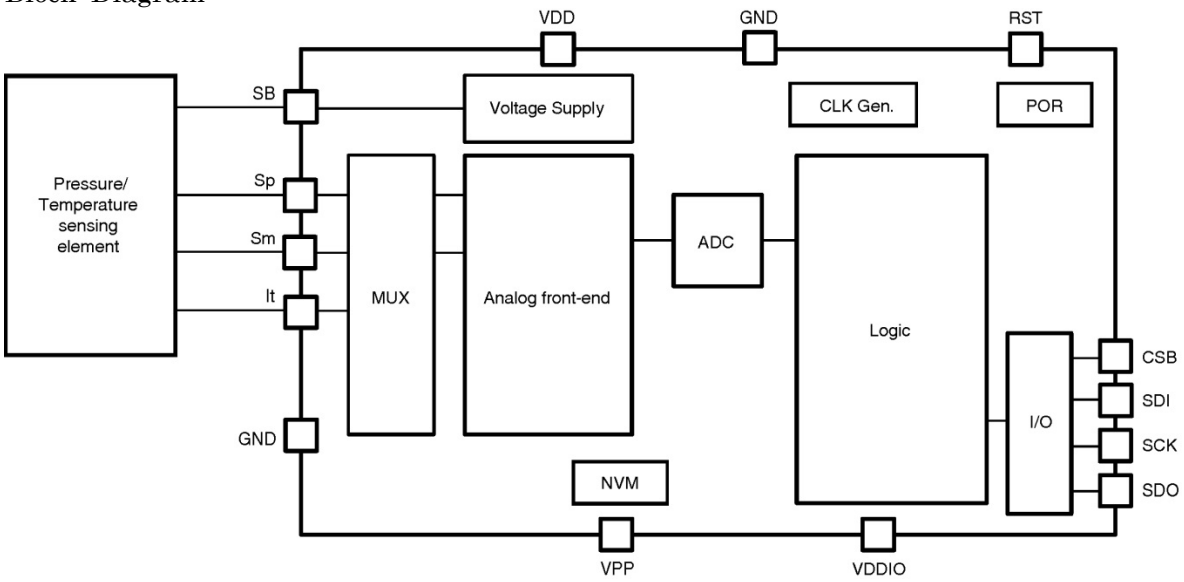
Span Voltage

$$\Delta Vo = \frac{\Delta R}{R} \cdot Vi$$

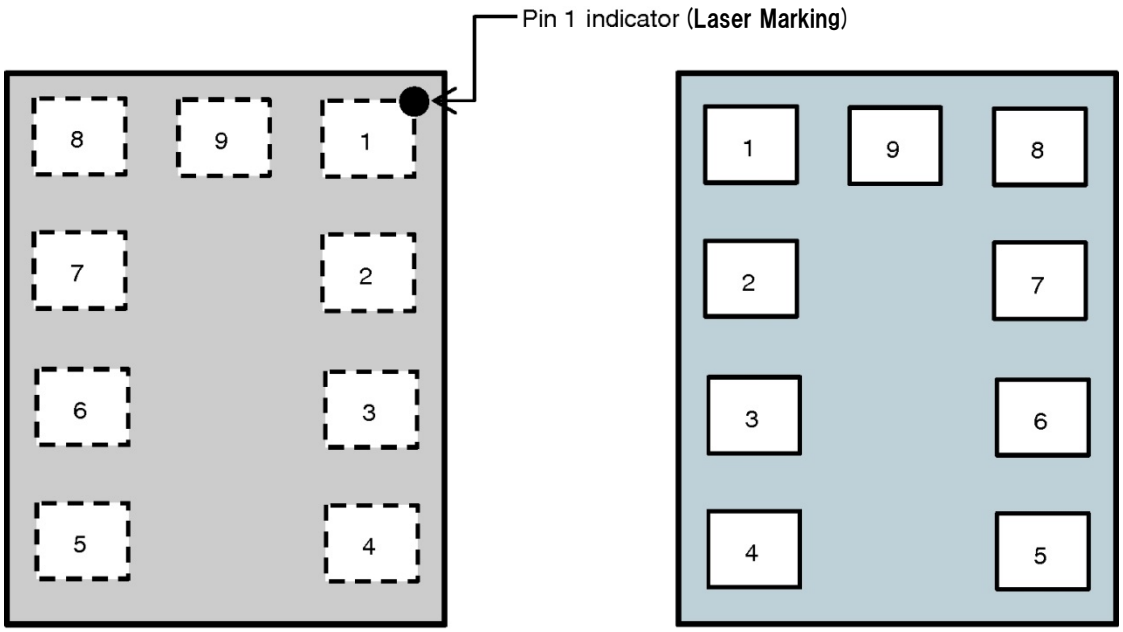
5. Usage

5.1. Connection

5.1.1. Block Diagram



5.1.2. Pin Description and Layout



Pin Description

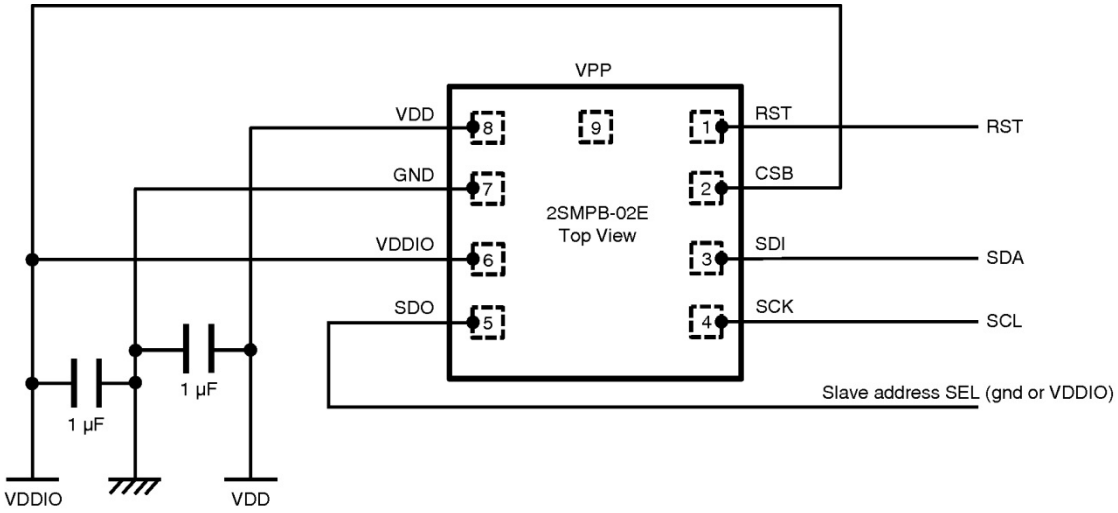
No.	Symbol	Description	
		SPI	I²C
1	RST	Asynchronous Reset *1)	
2	CSB	CSB	VDDIO
3	SDI	SDI/SDO	SDA
4	SCK	SCK	SCL
5	SDO	SDO	ADDR
6	VDDIO	Power Terminal for Digital IO	
7	GND	Ground Terminal	
8	VDD	Power Terminal	
9	VPP	NVM Writing Terminal *2)	

Note. *1) If you do not need the reset function, please just have the layout design of PCB of connecting both No.1 (RST) pin and No.7 (GND) pin into the ground of PCB.
Please refer "6.8 Reset Function" for the case of using the reset function.
*2) Pin 9 is only used internally in Omron. Please leave the pin disconnected.
If Pin 9 is connected with any other Pin electrically, the sensor will not work properly.

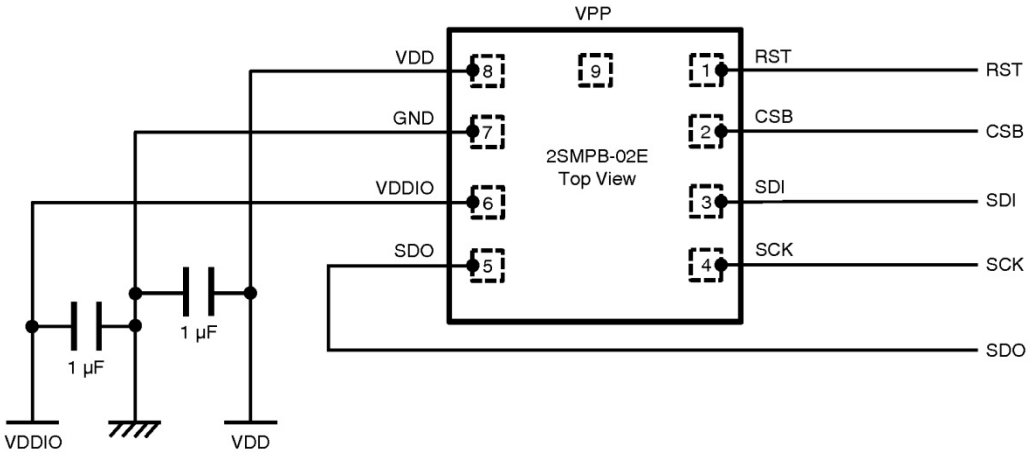
5.1.3. Typical Connection Diagram

(1) I²C mode

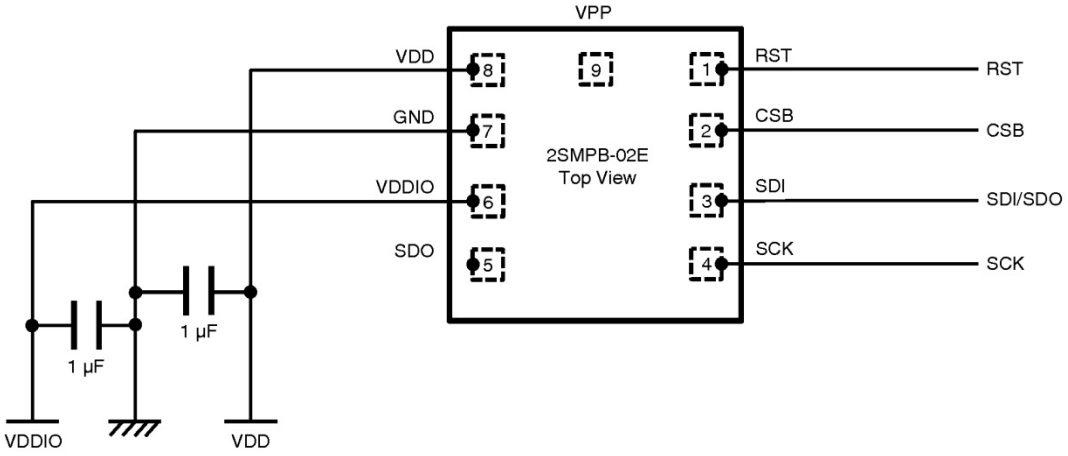
Corresponding to 100 Kbit/s (at Standard Mode), 400 Kbit/s (at Fast Mode) and 3.4 Mbit/s (at High Speed Mode)



(2) 4-wire SPI mode (Corresponding to 10 Mbit/s)



(3)3-wire SPI mode (Corresponding to 10 Mbit/s)

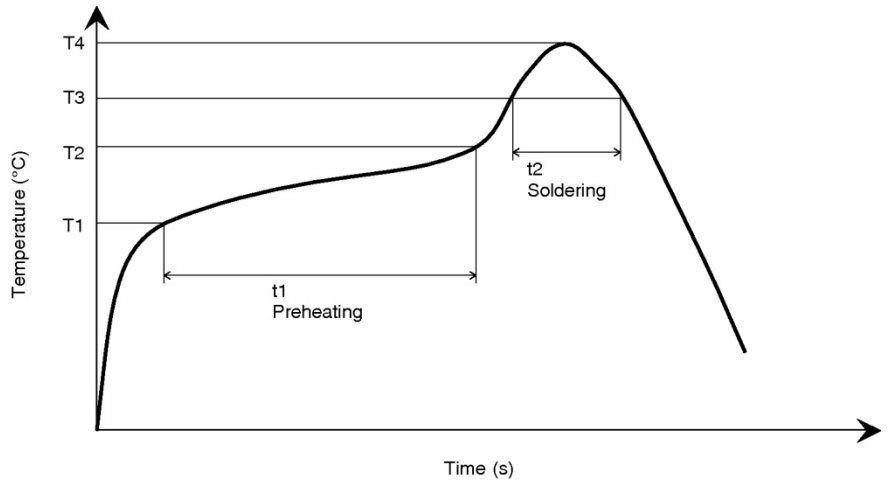


5.2. Recommended Soldering Method

■ **Soldering method** : Air Reflow (Max 2 times)

■ **Condition of Temperature** : Max.260 degree C, within 40 seconds

■ **Recommended Soldering Method** :



Temperature profile conditions of reflow soldering should set the temperature condition as shown in the below table and then confirm that actual conditions are met them in the table.

Item	Preheating (T1 to T2, t1)	Soldering (T3, t2)	Peak Value (T4)
Terminal	150 degree C to 200 degree C 60 s to 180 s.	217 degree C min. 60 s to 150 s.	260 degree C 20 s to 40 s.

- Since the pressure sensor chip is exposed to atmosphere, cleaning fluid shall not be allowed to enter inside the sensor's case.
- We recommend that it should be used the recommended mounting PAD dimensions for the land pattern.

5.3. Assembly Design Guide

5.3.1. Introduction

Omron's Digital Barometric Pressure Sensor (2SMPB-02 series) has high sensitivity and high accuracy with built-in MEMS sensor.

The process of detecting atmospheric pressure is as follows:

(1) Take in outside air through the pressure port

(2) Convert the value of barometric pressure to analog voltage output by MEMS sensor

(3) Furthermore, convert the analog voltage output to the digital signal

(4) Output the digital signal through the sensor terminal

The sensor output may be affected by the surrounding environment (temperature, humidity, stress, etc.) where the sensor is mounted, or the entry of foreign matter such as dust.

We will list the notes on the assembly of the sensor on the next pages.

As the degree of influence varies depending on the assembly design you need to confirm how your own assembly design affects the sensor output individually.

5.3.2. Notes on Assembly

① Assemble the sensor in a place where the measurement gas flows

If the sensor is not exposed to the gas to be measured from, the pressure of the gas can not be measured. If there is a narrow bottleneck that impedes flow between the sensor and the gas to be measured, it affects the frequency response. We recommend to set up our sensor close to the outside air.

Example) In the vicinity of the slot of SIM card or SD card, earphone jack, etc.

② Install a bypass capacitor to prevent conduction of noise from the power supply and GND.

Since the power supply and ground in the equipment are shared by various circuits, it can be a path of noise conduction. Barometric pressure sensor may be influenced by noise and the characteristics may fluctuate.

In order to prevent conduction of noise, we recommend decoupling by inserting a filter etc. Specifically, please connect externally a 1uF capacitor near the pad of the sensor between VDD-GND and VDDIO-GND.

③ Assemble the sensor in a place where ambient temperature is stable

The sensor output is affected by temperature fluctuation because of its measurement principle. The sensor can exclude the influence by temperature change.

However when a rapid temperature change, or a temperature change exceeding specifications occurs, it may affect the sensor characteristics.

Example) A place away from a heat generating element such as an MCU or a battery

④ Assemble the sensor in a place where the stress on the sensor is low and stable.

The sensor output is affected by external stress because of the structure of the sensor.

The sensor structure is devised so that the sensor is not easily affected by external stress.

However when a stress beyond our previous assumption is applied, there is a possibility that the sensor characteristics may be affected.

Stress may fluctuate not only on the mounted board but also when a force is applied from the lid side. Please do not push from the lid side, too.

Example) A place away from screws and fixtures to fix the mounted board. A place where a gap can be secured over the lid.

⑤ Avoid places that lead to sensor failure

Sensor is not waterproof, dustproof. In addition, condensation and freezing may cause malfunction. Please do not assemble in the following places.

・water is applied,

・condensation occurs,

・freezing occurs,

・foreign objects such as dust enter

6. Operations

6.1. Communication Mode

This sensor is corresponding to I²C and SPI communication.
Digital interface terminal functions for each communication mode are as below.

Communication mode	CSB	SDI	SCK	SDO	Remark
I²C	VDDIO	SDA	SCL	GND/VDDIO	SDO=GND→70h、SDO=VDDIO→56h
SPI 3 Wires	CSB	SDI/O	SCK	-	spi3w Register = 1
SPI 4 Wires	CSB	SDI	SCK	SDO	spi3w Register = 0

When changing the communication mode, also see Typical Connection Diagram section.

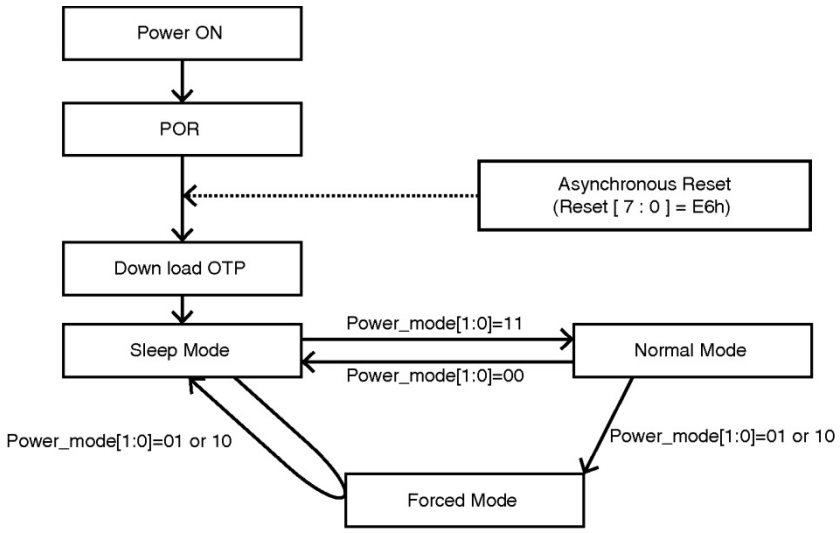
1) I²C mode becomes effective by pulling CSB up to VDDIO.
2) SPI mode becomes effective by pulling CSB down to GND.
3) Once CSB is pulled down, SPI mode would not be changed unless otherwise Power on Reset (POR) or Asynchronous Reset. Switching between SPI 3-Wire mode and SPI 4-Wire mode can be configured with the register value of "spi3w". Refer to IO_SETUP register section for more detail.
4) Default mode after POR or Asynchronous Reset will be I²C mode.

6.2. Power Mode

This sensor has three power modes and it can be switched by setting CTRL_MEAS register. Refer to the "CTRL_MEAS" register section for more detail.

- Sleep Mode
- Normal Mode
- Forced Mode

Transition diagram for each mode is as follows.

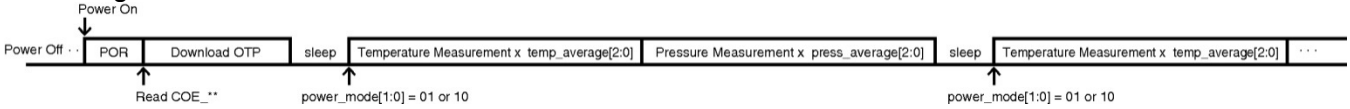


1) Sleep Mode (Power Reduction Mode)

No measurements are performed.
I²C/SPI interface and each register can be accessed even if the sensor is in sleep mode.

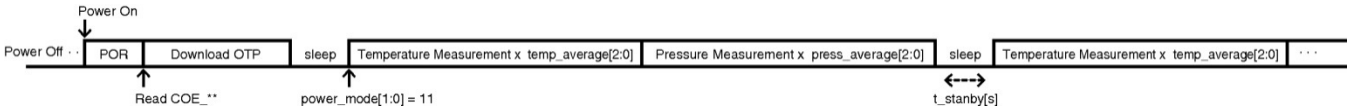
2) Forced Mode

In case of Forced Mode, a single measurement is performed. When the set up measurement is finished, the sensor returns to Sleep Mode after storing the measurement data to the register.



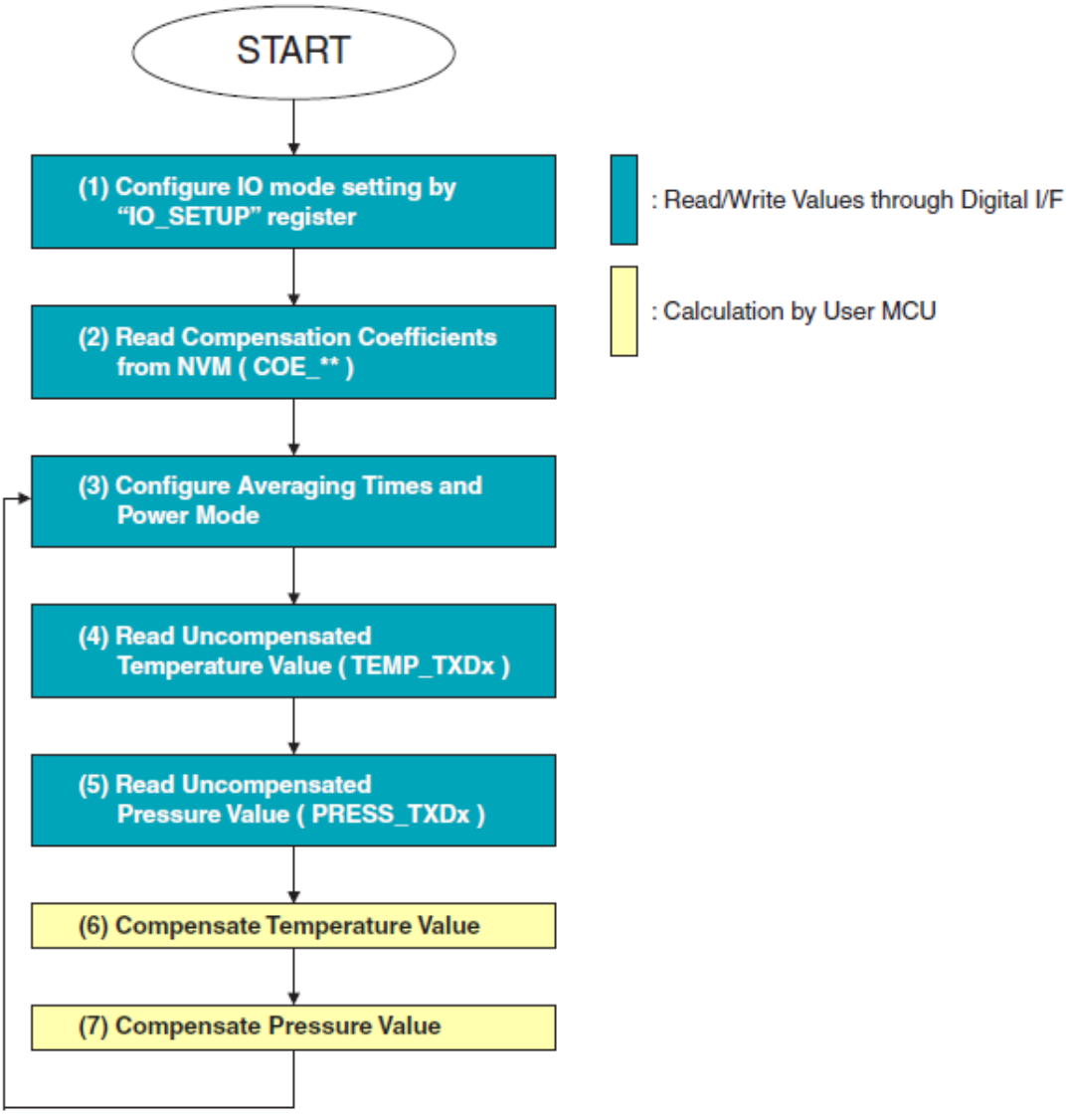
3) Normal Mode

In case of Normal Mode, the measurements are performed repeatedly between a measurement period and a standby period. The standby time can be configured by "t_stanby[2:0]" register. Be sure to consider that the data must be read from the master side after a Normal Mode.



6.3. Compensation of Pressure and Temperature

This section describes a typical measurement procedure and a calculation method after POR. This sensor has compensation coefficients in internal Non Volatile Memory (NVM). The compensated pressure can be calculated by using these values.



MSB of COE_** is sign bit.

(1) Configure IO mode setting. Refer to IO_SETUP register section for more detail.
(2) Read compensation coefficients which are stored in NVM. This procedure is sufficient just once after POR.
These values are used for a compensation calculation at the step (6) and (7).
(3) Configure averaging times and power mode. Refer to CTRL_MEAS register section for more detail.
(4) Read raw temperature data which are stored in TEMP_TXDx registers.
(5) Read raw pressure data which are stored in PRESS_TXDx registers.
(6) Compensated temperature can be calculated by using the below formula and the values of the step (2) and (4).

$$Tr = a0 + a1 \bullet Dt + a2 \bullet Dt^2$$

Tr Calculation Result of Temperature (Tr/256 = Temperature [degree C])
e.g.) If Tr Value is 6400 LSB,

$$\text{Temperature (degree C)} = \frac{\text{Tr Value (LSB)}}{\text{Scaling Factor}} = \frac{6400\ \text{LSB}}{256\ \text{LSB/degree C}} = 25.00\ \text{degree C}$$

Dt Raw Temperature Data [digit] (20-24 bits measurement value of TEMP_TXDx Reg.)
a0 Compensation Coefficient of PTAT (NVM resister: COE_a0_ex, COE_a0_0, COE_a0_1)
a1 Compensation Coefficient of PTAT (NVM resister: COE_a1_0, COE_a1_1)
a2 Compensation Coefficient of PTAT (NVM resister: COE_a2_0, COE_a2_1)

(7) Correction pressure without temperature compensation can be calculated by using the below formula and the values of the step (2) and (6).

$$\mathrm{Pr} = b00 + bt1 \bullet Tr + bp1 \bullet Dp + b11 \bullet Dp \bullet Tr + bt2 \bullet Tr^2 + bp2 \bullet Dp^2 + b12 \bullet Dp \bullet Tr^2 + b21 \bullet Dp^2 \bullet Tr + bp3 \bullet Dp^3$$

Pr	Calculation Result of Pressure [Pa]
Tr	Calculation Result of Temperature (Tr/256 = Temperature [degree C])
Dp	Raw Pressure Data [digit] (20-24 bits measurement value of PRESS_TXDx Reg.)
b00	Compensation Coefficient of Pressure (NVM resister: COE_b00_ex, COE_b00_0, COE_b00_1)
bt1	Compensation Coefficient of Pressure (NVM resister: COE_bt1_0, COE_bt1_1)
bp1	Compensation Coefficient of Pressure (NVM resister: COE_bp1_0, COE_bp1_1)
b11	Compensation Coefficient of Pressure (NVM resister: COE_b11_0, COE_b11_1)
bt2	Compensation Coefficient of Pressure (NVM resister: COE_bt2_0, COE_bt2_1)
bp2	Compensation Coefficient of Pressure (NVM resister: COE_bp2_0, COE_bp2_1)
b12	Compensation Coefficient of Pressure (NVM resister: COE_b12_0, COE_b12_1)
b21	Compensation Coefficient of Pressure (NVM resister: COE_b21_0, COE_b21_1)
bp3	Compensation Coefficient of Pressure (NVM resister: COE_bp3_0, COE_bp3_1)

- **How to get compensation coefficients**
Each compensation coefficients can be calculated by using the below formula and conversion factors.

$$K = A + \frac{S \times OTP}{32767} \quad \cdots \text{a1,a2,bt1,bt2,bp1,b11,bp2,b12,b21,bp3} \qquad K = \frac{OTP}{16} \cdots\cdots \text{a0,b00}$$

K	Conversion factor		OTP		
	A	S	23-16 bit	15-8 bit	7-0 bit
a1	-6.3E-03	4.3E-04	-	COE_a1_1	COE_a1_0
a2	-1.9E-11	1.2E-10	-	COE_a2_2	COE_a2_0
bt1	1.0E-01	9.1E-02	-	COE_bt1_1	COE_bt1_0
bt2	1.2E-08	1.2E-06	-	COE_bt2_1	COE_bt2_0
bp1	3.3E-02	1.9E-02	-	COE_bp1_1	COE_bp1_0
b11	2.1E-07	1.4E-07	-	COE_b11_1	COE_b11_0
bp2	-6.3E-10	3.5E-10	-	COE_bp2_1	COE_bp2_0
b12	2.9E-13	7.6E-13	-	COE_b12_1	COE_b12_0
b21	2.1E-15	1.2E-14	-	COE_b21_1	COE_b21_0
bp3	1.3E-16	7.9E-17	-	COE_bp3_1	COE_bp3_0

K	Conversion factor	OTP		
		19-12 bit	11-4 bit	3-0 bit
a0	Offset value (20Q16)	COE_a0_1	COE_a0_0	COE_a0_ex

b00	Offset value (20Q16)	COE_b00_1	COE_b00_0	COE_b00_ex

Example for "COE_**" bond

Example 1) In the case of 16 bits,

COE_a2 = (COE_a2_1 << 8) | COE_a2_0

* Treat 16th bit as MSB, complement of 2. Please note overflow.

Example 2) In the case of 20 bits,

COE_a2 = (COE_a2_1 << 8) | COE_a2_0

* Treat 20th bit as MSB, complement of 2. Please note overflow.

6.4. Implementing Register List

Register Name	Address		Length	R/W	Data								Discription	Initial
	I2C	SPI			bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0		
TEMP_TXD0	FCh	7Ch	8bit	R/-	t_txd0[7:0]								Temperature DATA[7:0] in 24bit	00h
TEMP_TXD1	FBh	7Bh	8bit	R/-	t_txd1[7:0]								Temperature DATA[15:8] in 24bit	00h
TEMP_TXD2	FAh	7Ah	8bit	R/-	t_txd2[7:0]								Temperature DATA[23:16] in 24bit (*)	00h
PRESS_TXD0	F9h	79h	8bit	R/-	p_txd0[7:0]								Pressure DATA[7:0] in 24bit	00h
PRESS_TXD1	F8h	78h	8bit	R/-	p_txd1[7:0]								Pressure DATA[15:8] in 24bit	00h
PRESS_TXD2	F7h	77h	8bit	R/-	p_txd2[7:0]								Pressure DATA[23:16] in 24bit (*)	00h
IO_SETUP	F5h	75h	8bit	R/W	t_stanby[2:0]			-		spi3_ sdim	-	spi3w	t_stanby[2:0] : Standby time setting spi3w : SPI mode setting (4 or 3 wire) spi3_sdim : Select output type of SDI terminal	00h
CTRL_MEAS	F4h	74h	8bit	R/W	temp_average[2:0]			press_average[2:0]			power_mode[1:0]		temp_average[2:0] : Temperature Averaging times press_average[2:0] : Pressure Averaging times power_mode[1:0] : Power mode setting	00h
DEVICE_STAT	F3h	73h	8bit	R/-	-	-	-	-	measure	-	-	otp_update	measure : Status of measurement otp_update : Status of OTP data access	00h
I2C_SET	F2h	72h	8bit	R/W	-	-	-	-	-	master_code[2:0]			Master code setting at I2C HS mode	01h
IIR_CNT	F1h	71h	8bit	R/W	-	-	-	-	-	filter[2:0]			IIR filter co-efficient setting	00h
RESET	E0h	60h	8bit	W	reset[7:0]								When inputting"E6h", a software reset will be occurred	00h
CHIP_ID	D1h	51h	8bit	R/-	chip_id[7:0]								CHIP_ID: 5Ch	5Ch
COE_b00_a0_ex	B8h	38h	8bit	R/-	b00[3:0] / a0[3:0]								Compensation Coefficient	-
COE_a2_0	B7h	37h	8bit	R/-	a2[7:0]								Compensation Coefficient	-
COE_a2_1	B6h	36h	8bit	R/-	a2[15:8]								Compensation Coefficient (*)	-
COE_a1_0	B5h	35h	8bit	R/-	a1[7:0]								Compensation Coefficient	-
COE_a1_1	B4h	34h	8bit	R/-	a1[15:8]								Compensation Coefficient (*)	-
COE_a0_0	B3h	33h	8bit	R/-	a0[11:4]								Compensation Coefficient	-
COE_a0_1	B2h	32h	8bit	R/-	a0[19:12]								Compensation Coefficient (*)	-
COE_bp3_0	B1h	31h	8bit	R/-	bp3[7:0]								Compensation Coefficient	-
COE_bp3_1	B0h	30h	8bit	R/-	bp3[15:8]								Compensation Coefficient (*)	-
COE_b21_0	AFh	2Fh	8bit	R/-	b21[7:0]								Compensation Coefficient	-
COE_b21_1	AEh	2Eh	8bit	R/-	b21[15:8]								Compensation Coefficient (*)	-
COE_b12_0	ADh	2Dh	8bit	R/-	b12[7:0]								Compensation Coefficient	-
COE_b12_1	ACh	2Ch	8bit	R/-	b12[15:8]								Compensation Coefficient (*)	-
COE_bp2_0	ABh	2Bh	8bit	R/-	bp2[7:0]								Compensation Coefficient	-
COE_bp2_1	AAh	2Ah	8bit	R/-	bp2[15:8]								Compensation Coefficient (*)	-
COE_b11_0	A9h	29h	8bit	R/-	b11[7:0]								Compensation Coefficient	-
COE_b11_1	A8h	28h	8bit	R/-	b11[15:8]								Compensation Coefficient (*)	-
COE_bp1_0	A7h	27h	8bit	R/-	bp1[7:0]								Compensation Coefficient	-
COE_bp1_1	A6h	26h	8bit	R/-	bp1[15:8]								Compensation Coefficient (*)	-
COE_bt2_0	A5h	25h	8bit	R/-	bt2[7:0]								Compensation Coefficient	-
COE_bt2_1	A4h	24h	8bit	R/-	bt2[15:8]								Compensation Coefficient (*)	-
COE_bt1_0	A3h	23h	8bit	R/-	bt1[7:0]								Compensation Coefficient	-
COE_bt1_1	A2h	22h	8bit	R/-	bt1[15:8]								Compensation Coefficient (*)	-
COE_b00_0	A1h	21h	8bit	R/-	b00[11:4]								Compensation Coefficient	-
COE_b00_1	A0h	20h	8bit	R/-	b00[19:12]								Compensation Coefficient (*)	-

(*) MSB of COE_** is sign bit.

TEMP(PRESS)_TXDx : Temperature and Pressure data : TXD0, TXD1 or TXD2

This sensor holds ADC data with 22 to 24 bits accuracy. It can be obtained as each 24 bits data. If there are redundant data, the low order positions will be filled by zero (0). The shaded regions as shown below are valid data area.

bit	24	23	22	...	5	4	3	2	1	Note
22 bits output	D21	D20	D19	...	D2	D1	D0	0	0	Temp/Press_ave=001
23 bits output	D22	D21	D20	...	D3	D2	D1	D0	0	Temp/Press_ave=010
24 bits output	D23	D22	D21	...	D4	D3	D2	D1	D0	Temp/Press_ave=011～111

※Dn(D23～D0) : Sensor Data ……The value of n bit (1 or 0)

※The raw measurement values are unsigned 24 bits values. The values need to do subtraction with 2^{23} at 24 bits output mode. Here is a programing example for Dt and Dp calculation.

$$Dt = ((TEMP\_TXD2) << 16) + ((TEMP\_TXD1) << 8) + (TEMP\_TXD0) - pow(2,23)$$

$$Dp = ((PRESS\_TXD2) << 16) + ((PRESS\_TXD1) << 8) + (PRESS\_TXD0) - pow(2,23)$$

IO_SETUP : IO SETUP Register

Register Name	I²C Addr.	SPI Addr.	Length	R/W	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	initial
IO_SETUP	F5h	75h	8bits	R/W	t_stanby[2:0]			-	-	spi3_sdim	-	spi3w	00h

bit7～5 t_stanby[2:0] : Standby time setting

000	001	010	011	100	101	110	111
1ms	5ms	50ms	250ms	500ms	1s	2s	4s

bit3～4 Reserved : keep these bits at 0

bit2 spi3_sdim : Select output type of SDI terminal
- 0 : Lo / Hi-Z output (Default)
- 1 : Lo / Hi output

bit1 Reserved : keep this bit at 0

bit0 spi3w : Change mode between SPI 4-wire and SPI 3-wire
- 0 : 4-wire (Default)
- 1 : 3-wire

CTRL_MEAS : Measurement Condition Control Register

Register Name	I²C Addr.	SPI Addr.	Length	R/W	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	initial
CTRL_MEAS	F4h	74h	8bits	R/W	temp_average[2:0]			press_average[2:0]			power_mode[1:0]		00h

bit7,6,5 temp_average[2:0] Averaging times setting for Temperature measurement (skip means no measurement.)

000	001	010	011	100	101	110	111
skip	1	2	4	8	16	32	64

bit4,3,2 press_average[2:0] Averaging times setting for Pressure measurement(skip means no measurement.)

000	001	010	011	100	101	110	111
skip	1	2	4	8	16	32	64

bit1,0 power_mode[1:0] Operation mode setting
- 00 : Sleep Mode
- 01,10 : Forced Mode
- 11 : Normal Mode

DEVICE_STAT : Device Status Register

Register Name	I²C Addr.	SPI Addr.	Length	R/W	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	initial
DEVICE_STAT	F3h	73h	8bits	R	-	-	-	-	measure	-	-	otp_update	00h

bit7～4 Reserved : keep these bits at 0

bit3 measure Device operation status. This value automatically changes.
- 0: Finish a measurement -- waiting for next measurement
- 1: On a measurement -- waiting for finishing the data store

bit2,1 Reserved : keep these bits at 0

bit0 otp_update The status of NVM data access. This value automatically changes.
- 0: No accessing NVM data
- 1: While accessing NVM data

I²C_SET : Master code setting

Register Name	I²C Addr.	SPI Addr.	Length	R/W	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	initial
I²C_SET	F2h	72h	8bits	R/W	-	-	-	-	-	master_code[2:0]			01h

bit7～3 Reserved : keep these bits at 0

bit2,1,0 master code[2:0] Master code setting at I²C high-speed mode.

000	001	010	011	100	101	110	111
08h	09h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh

IIR_CNT : IIR filter co-efficient setting Register

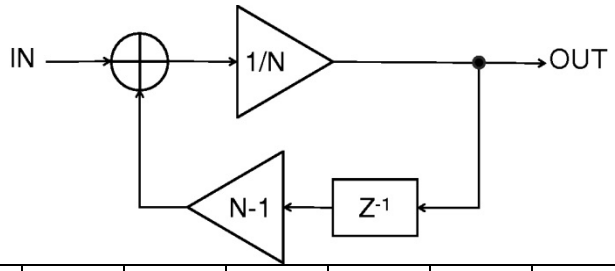
Register Name	I²C Addr.	SPI Addr.	Length	R/W	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	initial
IIR_CNT	F1h	71h	8bits	R/W	-	-	-	-	-	filter[2:0]			00h

bit7～3 Reserved : keep these bits at 0

bit2,1,0 filter[2:0] IIR filter co-efficient setting
Write access to this register address, IIR filter will be initialized.

Note. Initial setting of the IIR filter coefficient is “OFF”

.



000	001	010	011	100	101	110	111
OFF	N=2	N=4	N=8	N=16	N=32	N=32	N=32

RESET : Reset Control Register

Register Name	I²C Addr.	SPI Addr.	Length	R/W	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	initial
RESET	E0h	60h	8bits	W	reset[7:0]								00h

bit7～0 reset[7:0] When input "E6h", the software reset will be effective.
Except for that, nothing is to happen.

CHIP_ID : Chip ID Confirmation Register

Register Name	I²C Addr.	SPI Addr.	Length	R/W	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	initial
CHIP_ID	D1h	51h	8bits	R	chip_id[7:0]								5Ch

bit7～0 chip_id[7:0] 5Ch

6.5. I²C Protocol

(1) I²C Slave Address

The 2SMPB-02 module I²C slave address is shown below.

SDO	I²C Slave Address (7 bits)	Bit	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			Add[6]	Add[5]	Add[4]	Add[3]	Add[2]	Add[1]	Add[0]	R/W
High(1)	56h+R/W	Value	1	0	1	0	1	1	0	1/0
Low(0)	70h+R/W	Value	1	1	1	0	0	0	0	1/0

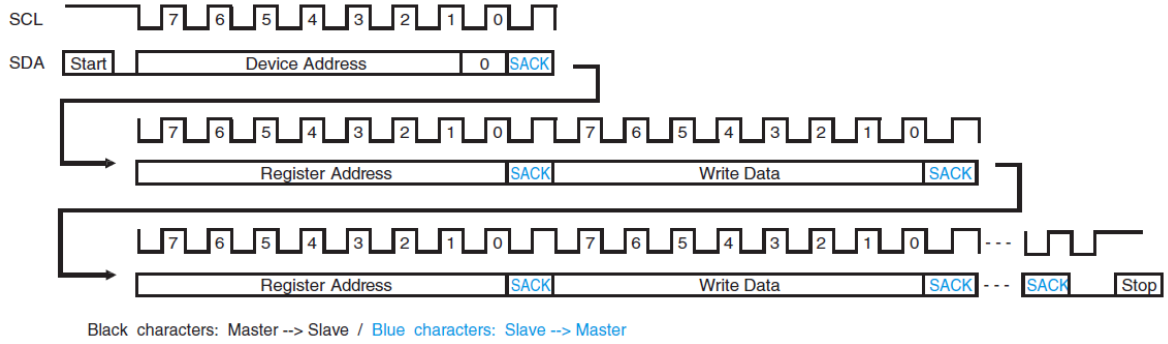
For example, in case of SDO=Low (0),
Write Access : Please set LSB of slave address as "0", then the address is E0h(1110_0000b). (70h<<1+WR(0))
Read Access : Please set LSB of slave address as "1", then the address is E1h(1110_0001b). (70h<<1+RD(1))

(2) I²C Access Protocol Examples

Symbol	Condition
START	START condition
STOP	STOP condition
SACK	Acknowledge by Slave
MACK	Acknowledge by Master
MNACK	Not Acknowledge by Master

(3) Register Write Access Protocol

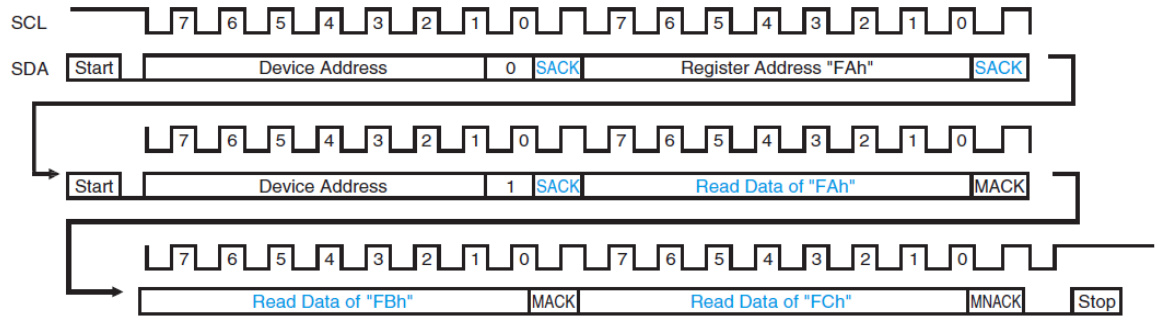
After the START condition, a Device Address is sent. This address is seven bits long followed by an eighth bit which is a data direction bit. A 'zero' indicates a transmission "WRITE". After that, the register address and the writing data shall be one set and it should be continuously transmitted until a STOP condition. A data transfer is always terminated by a STOP condition generated by the master.



Black characters: Master --> Slave / Blue characters: Slave --> Master

(4) Register Read Access Protocol

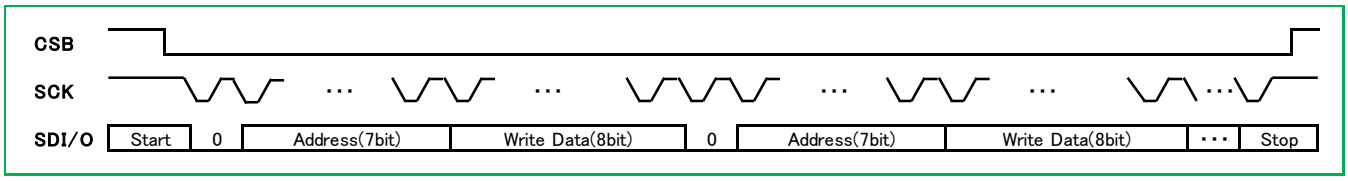
After a START condition, the Device Address with WRITE sign ("0") and Word Address intended to read a first data are transmitted. Next, "STOP–START" or "Re-START" condition are transmitted by the master. After that, Device Address with READ sign ("1") is transmitted by the master. Then, the slave will output the first data that is intended to read. In case of incrementing Register Address automatically, the slave will output the data repeatedly until NACK is input by the master. If Register Address becomes "FFh", please continue to output "FFh." Below example shows 3 bytes reading method from "FAh" register.



6.6. SPI Protocol

(1) SPI write

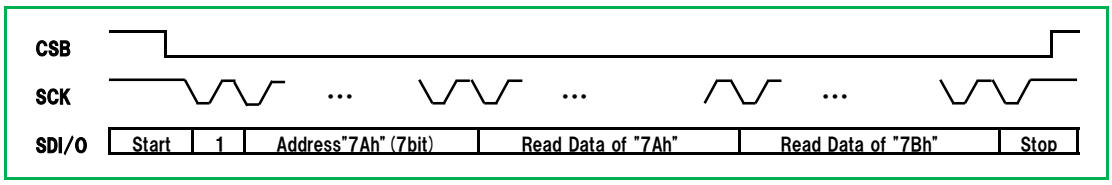
"SPI Write" needs to transmit the one set data of Register Address(Ctl.="0"+Address) and a writing data in the situation where CSB is "L". Two or more writing can be possible during CSB is "L". If CSB becomes "H", SPI communication will finish. (as well as I²C write)



(2) SPI read

First, "SPI read" needs to transmit Register Address(Ctl.="1"+Address) in a situation where CSB is "L". Next, the data of the requested register address will be output from SDO. (in case of 3-wire mode, the data will be output from SDI). After that, the register address is automatically incremented by one until CSB becomes "H", the device will output the data repeatedly. (as well as I²C read)

Below shows an example of the 2 bytes reading from "FAh" register.



6.7. Interface Specifications

(1) I²C timings

All timings apply to 100kbps（at Standard Mode）, 400kbps (at Fast Mode) and 3.4Mbps(at High Speed Mode).

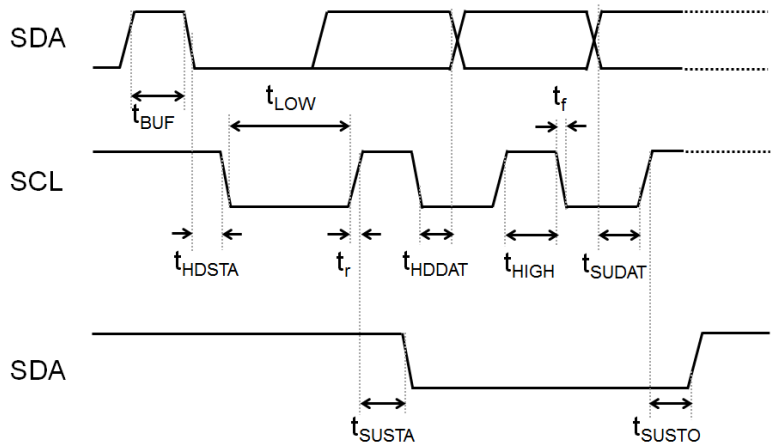
For I²C timings, the following abbreviations are used :

S&F Mode = Standard and Fast Mode

Cb = bus capacitance on SDI line

HS Mode = High Speed Mode

All other naming refers to I²C specification 2.1 (January 2000).



Undescribed items and symbols are compliant with the I²C specification.

Items	Symbol	Condition		min	typ	max	Units	Remark
SDI setup time	t_{SUDAT}	S&F Mode		160	-	-	ns	
		HS Mode	Vio=1.62 V	30	-	-	ns	
		HS Mode	Vio=1.2 V	55	-	-	ns	
SDI hold time	t_{HDDAT}	S&F Mode, Cb≦100 pF		80	-	-	ns	
		S&F Mode, Cb≦400 pF		90	-	-	ns	
		HS Mode, Cb≦100 pF	Vio=1.62 V	18	-	115	ns	
			Vio=1.2 V	25	-	140	ns	
		HS Mode, Cb≦400 pF	Vio=1.62 V	24	-	150	ns	
			Vio=1.2 V	45		170	ns	
SCK low pulse	t_{Low}	HS Mode, Cb≦100 pF	Vio=1.62 V	160	-	-	ns	
			Vio=1.2 V	210	-	-	ns	

(2) SPI timings

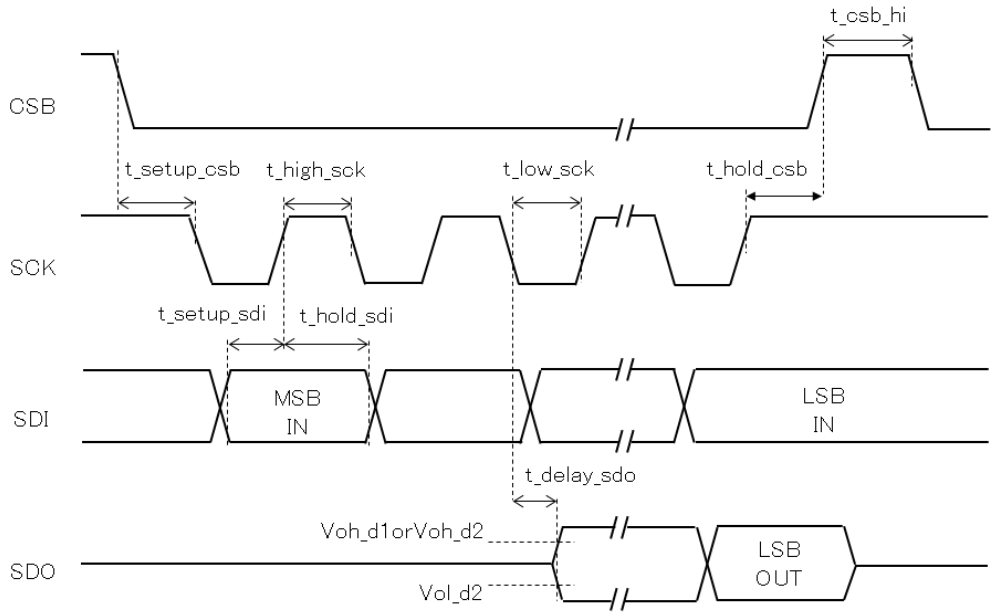
All timings are applied both to 4- and 3-wire SPI.

To reduce external noise in High-Z state, we recommend the following;

・In 4-wire mode, SDO terminal is pulled up to Vio via the resister.

・In 3-wire mode, SDI terminal is pulled up to Vio via the resister.

e.g.) Rpullup = 3.6kΩ @Vio=1.8V.



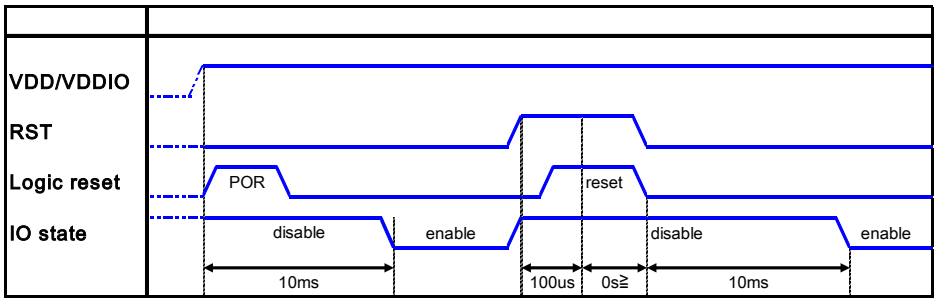
Items	Symbol	Condition	min	typ	max	Units	Remark
SCK frequency	f_spi		—	—	10	MHz	
SCK low pulse	t_low_sck		40	—	—	ns	
SCK high pulse	t_high_sck		40	—	—	ns	
SDI setup time	t_setup_sdi		20	—	—	ns	
SDI hold time	t_hold_sdi		20	—	—	ns	
SDO output delay	t_delay_sdo	Cb=25 pF,Vio=1.62 V min	—	—	30	ns	
		Cb=25 pF,Vio=1.2 V min	—	—	40	ns	
CSB setup time	t_setup_csb		40	—	—	ns	
CSB hold time	t_hold_csb		40	—	—	ns	
CSB_HI time	t_csb_hi		100	—	—	ns	

6.8. Reset Function

The sensor is capable of resetting the operation with "Asynchronous Reset Terminal (RST pin)".
The procedure is as follows:

① Input high voltage to RST pin. (100 us ≧)
② Turn off (input low voltage) and wait 10 ms.

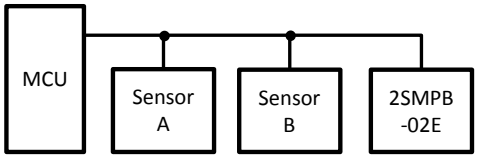
◆Reset sequence



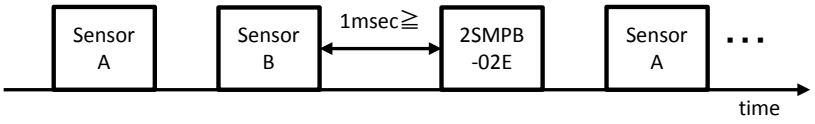
6.9. Recommended Conditions of Communication

In case that this sensor and other sensors are connected with a common bus line, if you use this sensor at a communication speed more than 400 kbit/s, after finishing the communication with other sensors, we recommend to provide 1 ms or more waiting time before starting the communication with this sensor in order to ensure a stable communication (see diagram below).

◆Typical connection diagram



◆Example of communication



7. Sample Source Code

The following is a sample source code of the 2SMPB-02E control in the case of using mbed NXP LPC1768.

7.1. 2SMPB02.h

```
 /** ******************************************************************************
 * ¥file 2SMPB02.h
 * ¥brief define 2SMPB-02 macro and constatnt values.
 *
 * ¥license {
 * This is free and unencumbered software released into the public domain.
 *
 * Anyone is free to copy, modify, publish, use, compile, sell, or distribute
 * this software, either in source code form or as a compiled binary,
 * for any purpose, commercial or non-commercial, and by any means.
 *
 * In jurisdictions that recognize copyright laws, the author or authors of this
 * software dedicate any and all copyright interest in the software to the
 * public domain. We make this dedication for the benefit of the public at large
 * and to the detriment of our heirs and successors. We intend this dedication
 * to be an overt act of relinquishment in perpetuity of all present and future
 * rights to this software under copyright law.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
 * AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
 * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 * }
 *
 * $Rev: 501:077bca797f3b $
 * $Id: 2SMPB02.h,v 077bca797f3b 2016/11/10 06:59:16 shinji_kuriyama $
 ********************************************************************************/
#ifndef _2SMPB02_H_
#define _2SMPB02_H_

//
// use configurations.
//

/** specify interface mode of the sensor */
// #define _2SMPB_CONFIG_SPI_INTERFACE 1
#define _2SMPB_CONFIG_I2C_INTERFACE 1

/** specify running mode of the sensor */
#undef _2SMPB02_NORMALMODE
// #define _2SMPB02_NORMALMODE          1

/** specify I2C speed to communicate the sensor */
#define _2SMPB02_FREQUENCY 400000L
```

```
/** 2SMPB-02 I2C Address (determined by pin connections) */
#define _2SMPB02_I2C_ADDR              ((uint8_t)0x56)
// #define _2SMPB02_I2C_ADDR              ((uint8_t)0x70)

//
// do not edit below lines.
//
#define _2SMPB02_I2C_ADDR_WR      (_2SMPB02_I2C_ADDR << 1)
#define _2SMPB02_I2C_ADDR_RD      (_2SMPB02_I2C_ADDR_WR | 0x01)

#if _2SMPB02_FREQUENCY > 400000L
    #define _2SMPB_I2CHS
#endif

// 2SMPB-02 register content values {{{1

/** 2SMPB-02 Hard wired value to get ID */
#define _2SMPB02_CHIP_ID1              ((uint8_t)0x5C)

/** Setup bits for 4sec period */
#define _2SMPB02_SETUP_STBY_4SEC ((uint8_t)0xE0)
/** Setup bits for 2sec period */
#define _2SMPB02_SETUP_STBY_2SEC ((uint8_t)0xC0)
/** Setup bits for 1sec period */
#define _2SMPB02_SETUP_STBY_1SEC ((uint8_t)0xA0)
/** Setup bits for 500msec period */
#define _2SMPB02_SETUP_STBY_500MSEC ((uint8_t)0x80)
/** Setup bits for 250msec period */
#define _2SMPB02_SETUP_STBY_250MSEC ((uint8_t)0x60)
/** Setup bits for 50msec period */
#define _2SMPB02_SETUP_STBY_50MSEC ((uint8_t)0x40)
/** Setup bits for 5msec period */
#define _2SMPB02_SETUP_STBY_5MSEC ((uint8_t)0x20)
/** Setup bits for 1msec period */
#define _2SMPB02_SETUP_STBY_1MSEC ((uint8_t)0x00)

/** 2SMPB-02 Status bit of measurement running */
#define _2SMPB02_STATUS_MEAS         ((uint8_t)8)

/** 2SMPB-02 Status bit of otp reading */
#define _2SMPB02_STATUS_OTP          ((uint8_t)1)

/** for I2C High speed mode */
#define _2SMPB02_I2CHS_MASTERCODE ((uint8_t)0x3)

/** IIR bits to set IIR off   */
#define _2SMPB02_IIR_OFF ((uint8_t)0)
/** IIR bits to set repeating IIR 2times */
#define _2SMPB02_IIR_2    ((uint8_t)1)
/** IIR bits to set repeating IIR 4times */
```

```
#define _2SMPB02_IIR_4    ((uint8_t)2)
/** IIR bits to set repeating IIR 8times */
#define _2SMPB02_IIR_8    ((uint8_t)3)
/** IIR bits to set repeating IIR 16times */
#define _2SMPB02_IIR_16   ((uint8_t)4)
/** IIR bits to set repeating IIR 32times */
#define _2SMPB02_IIR_32   ((uint8_t)5)

/** Reset hard-code for the reset register */
#define _2SMPB02_RESET_CODE ((uint8_t)0xE6)

#ifdef _2SMPB02_NORMALMODE
    /** 2SMPB-02 Operation mode to normal mode (run periodical) */
    #define _2SMPB02_CTRL_MODE            3
#else
    /** 2SMPB-02 Operation mode to force mode (run once) */
    #define _2SMPB02_CTRL_MODE            1
#endif


// 2SMPB-02 register definitions <!-- {{{1 -->
/** 2SMPB-02 registers for reading temperature digits */
#define _2SMPB02_TEMP_TXD2            ((uint8_t)0xFA)
/** 2SMPB-02 registers for reading pressure digits */
#define _2SMPB02_PRES_TXD2            ((uint8_t)0xF7)

/** 2SMPB-02 registers to control sensor running */
#define _2SMPB02_CTRL_MEAS_REG      ((uint8_t)0xF4)

/** 2SMPB-02 register to get hard wired id */
#define _2SMPB02_CHIP_ID1_REG       ((uint8_t)0xD1)

/** 2SMPB-02 register to set the sensor period and SPI settings */
#define _2SMPB02_SETUP_REG            ((uint8_t)0xF5)

/** 2SMPB-02 register to get the sensor running status */
#define _2SMPB02_STATUS_REG           ((uint8_t)0xF3)

/** 2SMPB-02 register to get the sensor running status */
#define _2SMPB02_I2CHS_REG            ((uint8_t)0xF2)

/** 2SMPB-02 register to get the sensor running status */
#define _2SMPB02_IIR_REG              ((uint8_t)0xF1)

/** 2SMPB-02 register to get the sensor running status */
#define _2SMPB02_RESET_REG            ((uint8_t)0xE0)

/** 2SMPB-02 registers to get calculation parameters */
#define _2SMPB02_PROM_START_ADDR   ((uint8_t)0xA0)

/** 2SMPB-02 length of calculation parameters in bytes */
```

```
#if defined(_2SMPB_PARAM_02E)
    #define _2SMPB_PROM_DATA_LEN      ((int32_t)25)
#endif

/*! power mode code = bit field in CTRL_MEAS */
#define _2SMPB02_MSK_MODE_POWER 0x03
#define _2SMPB02_VAL_MODE_SLEEP 0x00
#define _2SMPB02_VAL_MODE_FORCED 0x01
#define _2SMPB02_VAL_MODE_NORMAL 0x03

/*! average times bit fields of pressure in CTRL_MEAS */
#define _2SMPB02_MSK_MODE_P 0xE3
#define _2SMPB02_SFT_MODE_P 2
#define _2SMPB02_VAL_MODE_HIGHSPEED_P 0x04              /* 0bxxx0_01xx */
#define _2SMPB02_VAL_MODE_LOWPOWER_P 0x08               /* 0bxxx0_10xx */
#define _2SMPB02_VAL_MODE_STANDARD_P 0x10               /* 0bxxx1_00xx */
#define _2SMPB02_VAL_MODE_HIGHACCURACY_P 0x14           /* 0bxxx1_01xx */
#define _2SMPB02_VAL_MODE_ULTRAHIGHACCURACY_P 0x18      /* 0bxxx1_10xx */

/*! average times bit fields of temperature in CTRL_MEAS */
#define _2SMPB02_MSK_MODE_T 0x1F
#define _2SMPB02_SFT_MODE_T 5
#define _2SMPB02_VAL_MODE_HIGHSPEED_T 0x20              /* 0b001x_xxxx */
#define _2SMPB02_VAL_MODE_LOWPOWER_T 0x20               /* 0b001x_xxxx */
#define _2SMPB02_VAL_MODE_STANDARD_T 0x20               /* 0b001x_xxxx */
#define _2SMPB02_VAL_MODE_HIGHACCURACY_T 0x40           /* 0b010x_xxxx */
#define _2SMPB02_VAL_MODE_ULTRAHIGHACCURACY_T 0x40      /* 0b010x_xxxx */

/*! average times code */
#define _2SMPB02_VAL_MODE_HIGHSPEED ¥
    (_2SMPB02_VAL_MODE_HIGHSPEED_T | _2SMPB02_VAL_MODE_HIGHSPEED_P)
#define _2SMPB02_VAL_MODE_LOWPOWER ¥
    (_2SMPB02_VAL_MODE_LOWPWER_T | _2SMPB02_VAL_MODE_LOWPWER_P)
#define _2SMPB02_VAL_MODE_STANDARD ¥
    (_2SMPB02_VAL_MODE_STANDARD_T | _2SMPB02_VAL_MODE_STANDARD_P)
#define _2SMPB02_VAL_MODE_HIGHACCURACY ¥
    (_2SMPB02_VAL_MODE_HIGHACCURACY_T | _2SMPB02_VAL_MODE_HIGHACCURACY_P)
#define _2SMPB02_VAL_MODE_ULTRAHIGHACCURACY ¥
    (_2SMPB02_VAL_MODE_ULTRAHIGHACCURACY_T | ¥
     _2SMPB02_VAL_MODE_ULTRAHIGHACCURACY_P)

/** return codes */
#define _2SMPB02_OK 0
#define _2SMPB02_PENDING 1
#define _2SMPB02_NOT_INITIALIZED -1



/** struct _2SMPB02 {{{1
 *
 * * hold the calculation parameters
 * * hold virtual function to control I2C
```

```
 *
 */
typedef struct tag_2SMPB02 {
	/** otp data stream of calculation parameters */
	uint8_t otps[_2SMPB_PROM_DATA_LEN];

	/** otp data digits of calculation parameters */
	void* coef;

	/** read result of id1 register */
	uint8_t chip_id1;

	int32_t digitTemp, digitPres;
	double Po, Tx;
	uint8_t avg_p, avg_t, opmode;

	int (*write)(uint8_t reg, uint8_t* buf, int n);
	int (*read)(uint8_t reg, uint8_t* buf, int n);
	void (*wait_ms)(uint32_t usec);
} _2SMPB02;

// low level function definition
extern void _2SMPB02_spi_writes(_2SMPB02*, uint8_t, uint8_t*, int);
extern void _2SMPB02_spi_reads(_2SMPB02*, uint8_t, uint8_t*, int);

// function definition {{{1
extern int16_t convu16_s16(uint16_t src);
extern int32_t convu20_s32(uint32_t src);
extern int32_t convu24_s32(uint32_t src);

extern bool _2SMPB02_clear_coef(_2SMPB02*);
extern void _2SMPB02_setup(_2SMPB02*);
extern double _2SMPB02_convTx(_2SMPB02*, int32_t);
extern double _2SMPB02_get_pressure(_2SMPB02*, int32_t, double);

extern bool _2SMPB02_clear_coef_s64(_2SMPB02*);
extern void _2SMPB02_setup_s64(_2SMPB02*);
extern int16_t _2SMPB02_convTx_s64(_2SMPB02*, int32_t);
extern uint32_t _2SMPB02_get_pressure_s64(_2SMPB02*, int32_t, int16_t);

extern bool _2SMPB02_clear_coef_s32(_2SMPB02*);
extern void _2SMPB02_setup_s32(_2SMPB02*);
extern int16_t _2SMPB02_convTx_s32(_2SMPB02*, int32_t);
extern uint32_t _2SMPB02_get_pressure_s32(_2SMPB02*, int32_t, int16_t);

extern void _2SMPBQ_setup(_2SMPB02*);
extern bool _2SMPBQ_clear_coef_s64(_2SMPB02*);
extern int16_t _2SMPBQ_convTx_s64(_2SMPB02*, int32_t);
extern int32_t _2SMPBQ_get_pressure_s64(_2SMPB02*, int32_t, int16_t);

extern bool _2SMPBQ_clear_coef_dbl(_2SMPB02*);
```

```
extern double _2SMPBQ_convTx_dbl(_2SMPB02*, int32_t);
extern double _2SMPBQ_get_pressure_dbl(_2SMPB02*, int32_t, double);

extern void _2SMPBC_setup(_2SMPB02*);
extern bool _2SMPBC_clear_coef_s64(_2SMPB02*);
extern int16_t _2SMPBC_convTx_02e(_2SMPB02*, int32_t);
extern int32_t _2SMPBC_get_pressure_02e(_2SMPB02*, int32_t, int16_t);

extern bool _2SMPBC_clear_coef_dbl(_2SMPB02*);
extern double _2SMPBC_convTx_dbl(_2SMPB02*, int32_t);
extern double _2SMPBC_get_pressure_dbl(_2SMPB02*, int32_t, double);


extern bool _2SMPB02_find(_2SMPB02*);

extern int16_t _2SMPB02_get_cal_param(_2SMPB02*);

extern int16_t _2SMPB02_set_ctrl_meas(_2SMPB02*);
extern int32_t _2SMPB02_combine_txd(uint8_t*);
extern int32_t _2SMPB02_get_temp_data(_2SMPB02*);
extern int32_t _2SMPB02_get_press_data(_2SMPB02*);

#if defined(_2SMPB_MOCKBUILD_)
    #define oc_vrb(fmt, ...) printf("%s:"fmt, _func_, ##_VA_ARGS_)
    #define oc_dbg(fmt, ...) printf("%s:"fmt, _func_, ##_VA_ARGS_)
    #define oc_inf(fmt, ...) printf("%s:"fmt, _func_, ##_VA_ARGS_)
    #define oc_err(fmt, ...) printf("%s:"fmt, _func_, ##_VA_ARGS_)
#else
    #define oc_vrb(fmt, ...)
    #define oc_dbg(fmt, ...)
    #define oc_inf(fmt, ...)
    #define oc_err(fmt, ...)
#endif

#if defined(_cplusplus)
extern "C" {
#endif

extern bool _2SMPB02_init(_2SMPB02*, uint8_t, uint8_t, uint8_t);
extern int16_t _2SMPB02_run(_2SMPB02*);

#if defined(_cplusplus)
}
#endif


/*! convert u8 buffer to s32 (24bit) */
#define _2SMPB_MACRO_BYTES_U24(b, o) ¥
    (uint32_t)(((uint32_t)b[o] << 16) | ¥
                ((uint32_t)b[o + 1] << 8) | ¥
                (uint32_t)b[o + 2])
```

```
/*! convert u8 buffer to s16 */
#define _2SMPB_MACRO_BYTES_U16(b, o) ¥
     (uint32_t)(((uint16_t)b[o] << 8) | (uint16_t)b[o + 1])



/*************definition of parameters {{{1*********/
#if defined(_2SMPB_PARAM_02E)
     #define _2SMPB_DEVICE "OMRON_2SMPB-02E"
     #define _2SMPB02E_CA_A1_ -6.3E-03 // 4.3E-04
     #define _2SMPB02E_CA_A2_ -1.9E-11 // 1.2E-10
     #define _2SMPB02E_CA_BT1   1.0E-01 // 9.1E-02
     #define _2SMPB02E_CA_BT2   1.2E-08 // 1.2E-06
     #define _2SMPB02E_CA_BP1   3.3E-02 // 1.9E-02
     #define _2SMPB02E_CA_B11   2.1E-07 // 1.4E-07
     #define _2SMPB02E_CA_BP2 -6.3E-10 // 3.5E-10
     #define _2SMPB02E_CA_B12   2.9E-13 // 7.6E-13
     #define _2SMPB02E_CA_B21   2.1E-15 // 1.2E-14
     #define _2SMPB02E_CA_BP3   1.3E-16 // 7.9E-17

     #define _2SMPB02E_CS_A1_ 4.3E-04
     #define _2SMPB02E_CS_A2_ 1.2E-10
     #define _2SMPB02E_CS_BT1 9.1E-02
     #define _2SMPB02E_CS_BT2 1.2E-06
     #define _2SMPB02E_CS_BP1 1.9E-02
     #define _2SMPB02E_CS_B11 1.4E-07
     #define _2SMPB02E_CS_BP2 3.5E-10
     #define _2SMPB02E_CS_B12 7.6E-13
     #define _2SMPB02E_CS_B21 1.2E-14
     #define _2SMPB02E_CS_BP3 7.9E-17
#endif


/***************select parameters****************/
#if defined(_2SMPB_PARAM_02E)
     // #define   _2SMPB_VER OMRON_2SMPB_02E
     #define _2SMPBCUBIC_CA_BT2 _2SMPB02E_CA_BT2
     #define _2SMPBCUBIC_CS_BT2 _2SMPB02E_CS_BT2
     #define _2SMPBCUBIC_CA_BT1 _2SMPB02E_CA_BT1
     #define _2SMPBCUBIC_CS_BT1 _2SMPB02E_CS_BT1
     #define _2SMPBCUBIC_CA_BP1 _2SMPB02E_CA_BP1
     #define _2SMPBCUBIC_CS_BP1 _2SMPB02E_CS_BP1
     #define _2SMPBCUBIC_CA_B11 _2SMPB02E_CA_B11
     #define _2SMPBCUBIC_CS_B11 _2SMPB02E_CS_B11
     #define _2SMPBCUBIC_CA_BP2 _2SMPB02E_CA_BP2
     #define _2SMPBCUBIC_CS_BP2 _2SMPB02E_CS_BP2
     #define _2SMPBCUBIC_CA_B12 _2SMPB02E_CA_B12
     #define _2SMPBCUBIC_CS_B12 _2SMPB02E_CS_B12
     #define _2SMPBCUBIC_CA_B21 _2SMPB02E_CA_B21
     #define _2SMPBCUBIC_CS_B21 _2SMPB02E_CS_B21
```

```
    #define _2SMPBCUBIC_CA_BP3 _2SMPB02E_CA_BP3
    #define _2SMPBCUBIC_CS_BP3 _2SMPB02E_CS_BP3
    #define _2SMPBCUBIC_CA_A2_ _2SMPB02E_CA_A2_
    #define _2SMPBCUBIC_CS_A2_ _2SMPB02E_CS_A2_
    #define _2SMPBCUBIC_CA_A1_ _2SMPB02E_CA_A1_
    #define _2SMPBCUBIC_CS_A1_ _2SMPB02E_CS_A1_
#endif

#endif   // _2SMPB02_H_
// end of file {{{1
// vi:ft=c:et:nowrap:fdm=marker
```

7.2. 2SMPB_02E_int.c

```
/** ********************************************************************************
 * ¥file 2SMPB_02E_int.c
 * ¥brief define 2SMPB-02 functions around sensor hardware behaviors
 *
 * ¥license {
 * This is free and unencumbered software released into the public domain.
```

```
 *
 * Anyone is free to copy, modify, publish, use, compile, sell, or distribute
 * this software, either in source code form or as a compiled binary,
 * for any purpose, commercial or non-commercial, and by any means.
 *
 * In jurisdictions that recognize copyright laws, the author or authors of this
 * software dedicate any and all copyright interest in the software to the
 * public domain. We make this dedication for the benefit of the public at large
 * and to the detriment of our heirs and successors. We intend this dedication
 * to be an overt act of relinquishment in perpetuity of all present and future
 * rights to this software under copyright law.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
 * AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
 * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 * }
 *
 * WARNING:
 * This is only for evaluation purpose and
 * Users must be follow the calculation in datasheet.
 *
 * $Rev: f918b7b $
 * $Id: 2017-05-29 16:18:54 +0900 $
 ******************************************************************************/
#include <stdint.h>
#include <stdbool.h>
#include <math.h>

#if defined(DEBUG)
    #include <stdio.h>
#endif

#define _2SMPB_PARAM_02E 1
#include "2SMPB02.h"


/** selected functions <!-- {{{1 --> */
    #define _2SMPB_CALC_CUBIC

    #define _2SMPB_setup _2SMPBC_setup

        #define _2SMPB_clear_coef _2SMPBC_clear_coef_s64
        #define _2SMPB_convTx _2SMPBC_convTx_02e
        #define _2SMPB_get_pressure _2SMPBC_get_pressure_02e



/** local functions <!-- {{{1 --> */
```

```
static bool _2SMPB02_clear(_2SMPB02* obj);


/** buffer for sensor communications */
static uint8_t cmd[_2SMPB_PROM_DATA_LEN];


/** convu16_s16: helper: convert unsinged 16bit to signed <!-- {{{1 -->
 */
int16_t convu16_s16(uint16_t d) {
	int32_t ret;

	if ((d & 0x8000) != 0) {
		ret = (int32_t)d - 0x10000;
	} else {
		ret = d;
	}
	return ret;
}

/** convu20_s32: helper: convert unsinged 16bit to signed <!-- {{{1 -->
 */
int32_t convu20_s32(uint32_t d) {
	int32_t ret;

	if ((d & 0x80000) != 0) {
		ret = (int32_t)d - 0x100000;
	} else {
		ret = d;
	}
	return ret;
}

/** convu24_s32: helper: convert unsinged 16bit to signed <!-- {{{1 -->
 */
int32_t convu24_s32(uint32_t d) {
	int32_t ret;

	if ((d & 0x800000) != 0) {
		ret = (int32_t)d - 0x1000000;
	} else {
		ret = (int32_t)d;
	}
	return ret;
}

/** _2SMP02_init: initialize sensor data <!-- {{{1 -->
 *
 * **note** I2C speed must be normal or fast mode (<400kHz)
 *		during initializing the sensor.
 *
```

```
 * 1. initialize memory.
 * 2. software reset to the sensor.
 * 3. (optional) change I2C speed.
 * 4. find out the sensor on I2C/SPI.
 * 5. set IIR repeat to 16times.
 * 6. read calculate parameters via I2C/SPI.
*/
bool _2SMPB02_init(_2SMPB02* obj, uint8_t p, uint8_t t, uint8_t m) {
    // 1. initialize memory.
    _2SMPB02_clear(obj);
    obj->avg_p = p;         // oversampling of pressure
    obj->avg_t = t;         // oversampling of temperature
    obj->opmode = m;        // power mode: 'sleep', 'forced' or 'normal'

    // 2. software reset to the sensor.
    cmd[0] = _2SMPB02_RESET_CODE;
    obj->write(_2SMPB02_RESET_REG, cmd, 1);
    obj->wait_ms(10);   // 10ms

    #if defined(_2SMPB_CONFIG_I2C_INTERFACE) && defined(_2SMPB02_I2CHS)
        // 3. change I2C speed.
        cmd[0] = _2SMPB02_I2CHS_MASTERCODE;
        obj->write(_2SMPB02_I2CHS_REG, cmd, 1);

        // after this point, change your I2C Read/Write functions to
        // * send master code (0000 1??? <= ??? is _2SMPB02_I2CHS_MASTERCODE)
        // * change speed
        // * restore speed after stop condition.
    #endif

    // 4. find out the sensor by reading CHIPID & CHIPID2
    if (!_2SMPB02_find(obj)) {
        return true;
    }

    // 5. set IIR repeat to 16times.
    cmd[0] = _2SMPB02_IIR_16;
    obj->write(_2SMPB02_IIR_REG, cmd, 1);

    // 5. read calculation parameters SMPB02 calibparam structure
    _2SMPB02_get_cal_param(obj);

        // 7. start the sensor.
        _2SMPB02_set_ctrl_meas(obj);
    #endif

    return false;
}


/** _2SMPB02_clear: initialize sensor calculate parameters. <!-- {{{1 -->
```

```
 *
 */
bool _2SMPB02_clear(_2SMPB02* obj) {
	int i;

	for (i = 0; i < _2SMPB_PROM_DATA_LEN; i++) {
		obj->otps[i] = 0;
	}

	_2SMPB_clear_coef(obj);

	obj->chip_id1 = 0;

	obj->digitTemp = 0;
	obj->digitPres = 0;
	obj->Tx = 0.0;
	obj->Po = 0.0;
	obj->avg_p = 0;
	obj->avg_t = 0;
	obj->opmode = 0;

	return true;
}


/** _2SMPB02_find: find out the sensor <!-- {{{1 -->
 * by reading CHIP_ID_REG and CHIP_ID2_REG.
 */
bool _2SMPB02_find(_2SMPB02* obj) {
	int16_t err = 0;

	// read out Chip Id
	err += obj->read(_2SMPB02_CHIP_ID1_REG, cmd, 1);
	obj->chip_id1 = cmd[0];

	if (err != 0) {
		return false;
	}
	return obj->chip_id1 == _2SMPB02_CHIP_ID1;
}


/** _2SMPB02_get_cal_param: read all sensor calculate parameters <!-- {{{1 -->
 *
 */
int16_t _2SMPB02_get_cal_param(_2SMPB02* obj) {
	int16_t err = 0;

	err += obj->read(_2SMPB02_PROM_START_ADDR,
				obj->otps, _2SMPB_PROM_DATA_LEN);
```

```
    _2SMPB_setup(obj);
    return err;
}


/** _2SMPB02_set_ctrl_meas:  <!-- {{{1 -->
 *
 */
int16_t _2SMPB02_set_ctrl_meas(_2SMPB02* obj) {
    cmd[0] = (obj->avg_t << 5) | (obj->avg_p << 2) | obj->opmode;
    // oc_dbg("ctrl_meas: %x¥n", cmd[0]);
    return obj->write(_2SMPB02_CTRL_MEAS_REG, cmd, 1);
}


/** _2SMPB02_combine_txd: combine data stream to 24bit data <!-- {{{1 -->
 *
 */
int32_t _2SMPB02_combine_txd(uint8_t* data) {
    // "data" format
    // <--[0]-> <--[1]-> <--[2]->
    // <--msb-> <--lsb-> <-xlsb->
    // 76543210 76543210 76543210
    // dddddddd dddddddd dddddddd   d: valid_data, 0: unimplemented
    // <-----20bit---------->      average==1
    // <-----21bit----------->     average==2
    // <-----22bit------------>    average==3
    // <-----23bit-------------> average==4
    // <-----24bit--------------->average>=5
    uint32_t ud;
    int32_t ret;

    ud = (((uint32_t)data[0] << 16) |
          ((uint32_t)data[1] << 8) |
           (uint32_t)data[2]);
    ret = (int32_t)ud - 0x800000;
    // oc_dbg("combine_txd: %x = %d¥n", ret, ret);
    return ret;
}


/** _2SMPB02_get_temp_data: Get temperature data <!-- {{{1 -->
 *
 */
int32_t _2SMPB02_get_temp_data(_2SMPB02* obj) {
    int16_t err = 0;

    err += obj->read(_2SMPB02_TEMP_TXD2, cmd, 3);
    if (err != 0) {return 0;}

    return _2SMPB02_combine_txd(cmd);
```

```
}


/** _2SMPB02_get_press_data: Get pressure digit <!-- {{{1 -->
 *
 */
int32_t _2SMPB02_get_press_data(_2SMPB02* obj) {
    int32_t err = 0;

    err += obj->read(_2SMPB02_PRES_TXD2, cmd, 3);
    if (err != 0) {return 0;}

    return _2SMPB02_combine_txd(cmd);
}


/** _2SMPB02_is_running: Get sensor status <!-- {{{1 -->
 *
 */
bool _2SMPB02_is_running(_2SMPB02* obj) {
    int32_t err = 0;

    err += obj->read(_2SMPB02_STATUS_REG, cmd, 1);
    if (err != 0) {return true;}

    if (cmd[0] & (_2SMPB02_STATUS_MEAS | _2SMPB02_STATUS_OTP)) {
        return true;
    }
    return false;
}


/** _2SMPB02_run: The total flow to get pressure data in [Pa] <!-- {{{1 -->
 *
 */
int16_t _2SMPB02_run(_2SMPB02* obj) {
    if (obj->coef == NULL) {
        obj->digitTemp = 0;
        obj->digitPres = 0;
        obj->Tx = 0;
        obj->Po = 0;
        return _2SMPB02_NOT_INITIALIZED;
    }

    obj->digitTemp = _2SMPB02_get_temp_data(obj);
    obj->digitPres = _2SMPB02_get_press_data(obj);

    obj->Tx = _2SMPB_convTx(obj, obj->digitTemp);
    obj->Po = _2SMPB_get_pressure(obj, obj->digitPres, obj->Tx);

    #if !defined(_2SMPB02_NORMALMODE)
```

```
            _2SMPB02_set_ctrl_meas(obj);
    #endif
    return _2SMPB02_OK;
}


/** compensation parameter of the 2SMPB series Quadratic compensation,
*/
typedef struct tag_2SMPBC_COEF {
    /** initialized calculation parameters */
    bool is_initialized;

    /** otp data by multi bytes */
    uint32_t B00, A0_;              // 20Q4
    uint16_t BT1, BP1;
    uint16_t BT2, B11, BP2;
    uint16_t B12, B21, BP3;
    uint16_t A1_, A2_;

    /** otp data by multi bytes */
    int32_t b00, a0_;
    int32_t bt1, bp1;
    int64_t bt2;
    int32_t b11, bp2;
    int32_t b12, b21, bp3;
    int32_t a1_, a2_;
} _2SMPBC_COEF;

/** need to change multi instances for multi sensors */
static _2SMPBC_COEF _2smpbc_coef = {
    .is_initialized = false
};

/** compensation parameter values: _A_ is center, _S_ is range. */

#define _2SMPBCUBIC_OFFSET_B00 0
#define _2SMPBCUBIC_OFFSET_BT1 2
#define _2SMPBCUBIC_OFFSET_BT2 4
#define _2SMPBCUBIC_OFFSET_BP1 6
#define _2SMPBCUBIC_OFFSET_B11 8
#define _2SMPBCUBIC_OFFSET_BP2 10
#define _2SMPBCUBIC_OFFSET_B12 12
#define _2SMPBCUBIC_OFFSET_B21 14
#define _2SMPBCUBIC_OFFSET_BP3 16
#define _2SMPBCUBIC_OFFSET_A0_ 18
#define _2SMPBCUBIC_OFFSET_A1_ 20
#define _2SMPBCUBIC_OFFSET_A2_ 22
#define _2SMPBCUBIC_OFFSET_B00_A0_EX    24


/** local funtions */
```

```
static void _otp2coef_02e(_2SMPB02* obj);


/** <!-- _clear_coef_s64 {{{1 --> combine 8 digits to word variables
 */
bool _2SMPBC_clear_coef_s64(_2SMPB02* obj) {
	_2SMPBC_COEF* coe;
	obj->coef = (void*)(coe = &_2smpbc_coef);

	coe->B00 = 0;
	coe->BT1 = 0;
	coe->BT2 = 0;
	coe->BP1 = 0;
	coe->B11 = 0;
	coe->BP2 = 0;
	coe->A0_ = 0;
	coe->A1_ = 0;
	coe->A2_ = 0;

	coe->b00 = 0;
	coe->bt1 = 0;
	coe->bt2 = 0;
	coe->bp1 = 0;
	coe->b11 = 0;
	coe->bp2 = 0;
	coe->a0_ = 0;
	coe->a1_ = 0;
	coe->a2_ = 0;

	coe->is_initialized = false;
	return false;
}



/** _joinotp: combine 8 digits to word variables <!-- {{{1 -->
 *
 */
static void _joinotp(_2SMPB02* obj) {
	_2SMPBC_COEF* coe = (_2SMPBC_COEF*)obj->coef;
	uint8_t* buf = obj->otps;
	uint8_t ex;

	coe->B00 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_B00);
	coe->BT1 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_BT1);
	coe->BT2 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_BT2);
	coe->BP1 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_BP1);
	coe->B11 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_B11);
	coe->BP2 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_BP2);
	coe->B12 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_B12);
	coe->B21 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_B21);
```

```
	coe->BP3 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_BP3);
	coe->A0_ = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_A0_);
	coe->A1_ = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_A1_);
	coe->A2_ = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_A2_);

	// join extra 4bits.
	ex = buf[_2SMPBCUBIC_OFFSET_B00_A0_EX];
	coe->A0_ = (coe->A0_ << 4) | (ex & 0x0F);                 // 20Q4
	coe->B00 = (coe->B00 << 4) | ((ex >> 4) & 0x0F);       // 20Q4

	oc_dbg("B00: %x(uint)¥n", coe->B00);
	oc_dbg("BT1: %x(uint)¥n", coe->BT1);
	oc_dbg("BT2: %x(uint)¥n", coe->BT2);
	oc_dbg("BP1: %x(uint)¥n", coe->BP1);
	oc_dbg("B11: %x(uint)¥n", coe->B11);
	oc_dbg("BP2: %x(uint)¥n", coe->BP2);
	oc_dbg("B12: %x(uint)¥n", coe->B12);
	oc_dbg("B21: %x(uint)¥n", coe->B21);
	oc_dbg("BP3: %x(uint)¥n", coe->BP3);
	oc_dbg("A0_: %x(uint), %d¥n", coe->A0_, coe->A0_);
	oc_dbg("A1_: %x(uint)¥n", coe->A1_);
	oc_dbg("A2_: %x(uint)¥n", coe->A2_);

	coe->is_initialized = true;
}



/** _2SMPBC_setup: calcurate parameters from I2C data stream <!-- {{{1 -->
 *
 */
void _2SMPBC_setup(_2SMPB02* obj) {
	/** TODO: multiple DUTs */
	obj->coef = (void*)&_2smpbc_coef;
	_joinotp(obj);
	_otp2coef_02e(obj);
}



/** _otp2coef_02e: convert 16bit words to float parameters <!-- {{{1 -->
 *
 */
static void _otp2coef_02e(_2SMPB02* obj) {
	_2SMPBC_COEF* coe = (_2SMPBC_COEF*)obj->coef;

	if (!coe || !coe->is_initialized) {
		/** (coe->B00 or etc) is not valid in this time. */
		return;
	}
```

```
	coe->b00 = convu20_s32(coe->B00);                                  // 20Q4
	coe->bt1 = 2982L * (int64_t)convu16_s16(coe->BT1) + 107370906L;    // 28Q15
	coe->bt2 = 329854L * (int64_t)convu16_s16(coe->BT2) + 108083093L; // 34Q38

	coe->bp1 = 19923L * (int64_t)convu16_s16(coe->BP1) + 1133836764L; // 31Q20
	coe->b11 = 2406L * (int64_t)convu16_s16(coe->B11) + 118215883L;    // 28Q34
	coe->bp2 = 3079L * (int64_t)convu16_s16(coe->BP2) - 181579595L;    // 29Q43

	coe->b12 = 6846L * (int64_t)convu16_s16(coe->B12) + 85590281L;     // 29Q53
	coe->b21 = 13836L * (int64_t)convu16_s16(coe->B21) + 79333336L;    // 29Q60
	coe->bp3 = 2915L * (int64_t)convu16_s16(coe->BP3) + 157155561L;    // 28Q65

	coe->a0_ = convu20_s32(coe->A0_);                                  // 20Q4
	coe->a1_ = 3608L * (int32_t)convu16_s16(coe->A1_) - 1731677965L;   // 31Q23
	coe->a2_ = 16889L * (int32_t)convu16_s16(coe->A2_) - 87619360L;    // 30Q47

	oc_dbg("b00,bt1,bt2: %10d, %10d, %10d¥n",
		coe->b00, coe->bt1, coe->bt2);
	oc_dbg("bp1,b11,bp2: %10d, %10d, %10d¥n",
		coe->bp1, coe->b11, coe->bp2);
	oc_dbg("b12,b21,bp3: %10d, %10d, %10d¥n",
		coe->b12, coe->b21, coe->bp3);
	oc_dbg("a0_,a1_,a2_: %10d, %10d, %10d¥n",
		coe->a0_, coe->a1_, coe->a2_);
}

/** _2SMPBC_convTx_02e: convert temperature digit to 256 degC <!-- {{{1 -->
 *
 */
int16_t _2SMPBC_convTx_02e(_2SMPB02* obj, int32_t dt) {
	int16_t ret;
	int64_t wk1, wk2;

	_2SMPBC_COEF* coe = (_2SMPBC_COEF*)obj->coef;
	if (!coe || !coe->is_initialized) {return 0.0;}

	// wk1: 60Q4                                                       // bit size
	wk1 = ((int64_t)coe->a1_ * (int64_t)dt);             // 31Q23+24-1=54 (54Q23)
	wk2 = ((int64_t)coe->a2_ * (int64_t)dt) >> 14;    // 30Q47+24-1=53 (39Q33)
	wk2 = (wk2 * (int64_t)dt) >> 10;                        // 39Q33+24-1=62 (52Q23)
	wk2 = ((wk1 + wk2) / 32767) >> 19;                      // 54,52->55Q23   (20Q04)
	ret = (int16_t)((coe->a0_ + wk2) >> 4);              // 21Q4 -> 17Q0
	return ret;
}


/** _2SMPBC_get_pressure_02e: convert digit data to pressure[Pa] <!-- {{{1 -->
 *
 */
int32_t _2SMPBC_get_pressure_02e(_2SMPB02* obj, int32_t dp, int16_t tx) {
	int32_t ret;
```

```
    int64_t wk1, wk2, wk3;

    _2SMPBC_COEF* coe = (_2SMPBC_COEF*)obj->coef;
    if (!coe || !coe->is_initialized) {return 0.0;}

    // wk1 = 48Q16                                          // bit size
    wk1 = ((int64_t)coe->bt1 * (int64_t)tx);             // 28Q15+16-1=43 (43Q15)
    wk2 = ((int64_t)coe->bp1 * (int64_t)dp) >> 5;        // 31Q20+24-1=54 (49Q15)
    wk1 += wk2;                                          // 43,49->50Q15

    wk2 = ((int64_t)coe->bt2 * (int64_t)tx) >> 1;        // 34Q38+16-1=49 (48Q37)
    wk2 = (wk2 * (int64_t)tx) >> 8;                      // 48Q37+16-1=63 (55Q29)
    wk3 = wk2;                                           // 55Q29

    wk2 = ((int64_t)coe->b11 * (int64_t)tx) >> 4;        // 28Q34+16-1=43 (39Q30)
    wk2 = (wk2 * (int64_t)dp) >> 1;                      // 39Q30+24-1=62 (61Q29)
    wk3 += wk2;                                          // 55,61->62Q29

    wk2 = ((int64_t)coe->bp2 * (int64_t)dp) >> 13;       // 29Q43+24-1=52 (39Q30)
    wk2 = (wk2 * (int64_t)dp) >> 1;                      // 39Q30+24-1=62 (61Q29)
    wk3 += wk2;                                          // 62,61->63Q29

    wk1 += wk3 >> 14;                                    // Q29 >> 14 -> Q15

    wk2 = ((int64_t)coe->b12 * (int64_t)tx);             // 29Q53+16-1=45 (45Q53)
    wk2 = (wk2 * (int64_t)tx) >> 22;                     // 45Q53+16-1=61 (39Q31)
    wk2 = (wk2 * (int64_t)dp) >> 1;                      // 39Q31+24-1=62 (61Q30)
    wk3 = wk2;                                           // 61Q30

    wk2 = ((int64_t)coe->b21 * (int64_t)tx) >> 6;        // 29Q60+16-1=45 (39Q54)
    wk2 = (wk2 * (int64_t)dp) >> 23;                     // 39Q54+24-1=62 (39Q31)
    wk2 = (wk2 * (int64_t)dp) >> 1;                      // 39Q31+24-1=62 (61Q20)
    wk3 += wk2;                                          // 61,61->62Q30

    wk2 = ((int64_t)coe->bp3 * (int64_t)dp) >> 12;       // 28Q65+24-1=51 (39Q53)
    wk2 = (wk2 * (int64_t)dp) >> 23;                     // 39Q53+24-1=62 (39Q30)
    wk2 = (wk2 * (int64_t)dp);                           // 39Q30+24-1=62 (62Q30)
    wk3 += wk2;                                          // 62,62->63Q30

    wk1 += wk3 >> 15;     // Q30 >> 15 = Q15

    wk1 /= 32767L;
    wk1 >>= 11;              // Q15 >> 7 = Q4
    wk1 += coe->b00;         // Q4 + 20Q4
    wk1 >>= 4;               // 28Q4 -> 24Q0

    ret = (int32_t)wk1;
    return ret;
}

// end of file {{{1
```

```
#if !defined(_2SMPB_MOCK_DEBUG)

static _2SMPB02 inst;   // Sensor struct

#if defined(_2SMPB_WITH_MBED) && defined(_2SMPB_CONFIG_I2C_INTERFACE)
    #include "mbed.h"

    I2C i2c(p28, p27);

    Serial pc(USBTX, USBRX);
    DigitalOut gpio1(p20);

    static uint8_t mbed_cmd[_2SMPB_PROM_DATA_LEN];

    // low level function definition
    int _2SMPB_i2c_write(uint8_t addr, uint8_t* buf, int n) {
        return i2c.write(addr, (const char*)buf, n);
    }

    int _2SMPB_i2c_read(uint8_t addr, uint8_t* buf, int n) {
        return i2c.read(addr, (char*)buf, n);
    }

    int _2SMPB_i2c_writes(uint8_t reg, uint8_t* buf, int n) {
        int i, err = 0;

        mbed_cmd[0] = reg;
        for (i = 0; i < n; i++) {
            mbed_cmd[1 + i] = buf[i];
        }
        err += _2SMPB_i2c_write(_2SMPB02_I2C_ADDR_WR, mbed_cmd, n + 1);
        return err;
    }

    int _2SMPB_i2c_reads(uint8_t reg, uint8_t* buf, int n) {
        int err = 0;

        mbed_cmd[0] = reg;
        err += _2SMPB_i2c_write(_2SMPB02_I2C_ADDR_WR, mbed_cmd, 1);
        err += _2SMPB_i2c_read(_2SMPB02_I2C_ADDR_RD, buf, n);
        return err;
    }

#elif defined(_2SMPB_WITH_MBED) && !defined(_2SMPB_CONFIG_I2C_INTERFACE)
    #include "mbed.h"

    #if defined(_2SMPB_WITH_MBED_SPI1)
        SPI spi(p5, p6, p7);
        DigitalOut gpio_ss(p8);
    #elif defined(_2SMPB_WITH_MBED_SPI2)
```

```
        SPI spi(p11, p12, p13);
        DigitalOut gpio_ss(p14);
    #else
        #error please set 1 or 2 to _2SMPB_WITH_MBED for select peripheral.
    #endif

    Serial pc(USBTX, USBRX);
    DigitalOut gpio1(p20);

    void _2SMPB_spi_setss(bool hi_low) {
        // hi_low: 1 => hi, 0 => low.
        gpio_ss = hi_low;
    }

    uint8_t _2SMPB_spi_write(uint8_t val) {
        return spi.write(val);
    }

    int _2SMPB_spi_writes(uint8_t reg, uint8_t* buf, int n) {
        int i;

        _2SMPB_spi_setss(0);
        _2SMPB_spi_write(reg & 0x7F);
        for (i = 0; i < n; i++) {
            _2SMPB_spi_write(buf[i]);
        }
        wait_ms(1);
        _2SMPB_spi_setss(1);
        return 0;
    }

    int _2SMPB_spi_reads(uint8_t reg, uint8_t* buf, int n) {
        int i;

        _2SMPB_spi_setss(0);
        _2SMPB_spi_write(reg);
        for (i = 0; i < n; i++) {
            buf[i] = _2SMPB_spi_write(0xff);
        }
        wait_ms(1);
        _2SMPB_spi_setss(1);
        return 0;
    }

#else
    void exit();      // exit() is a mbed function.
#endif

#if defined(_2SMPB_WITH_MBED)
    void _wait_ms(uint32_t usec) {
        wait_us((int)usec * 1000);
```

```
    }
#endif


int main() {
    // select functions by your system
    #if 0
        inst.read = your_read_function;
        inst.write = your_write_function;
        inst.wait_ms = your_wait_micro_sec_function;

    #elif defined(_2SMPB_WITH_MBED) && defined(_2SMPB_CONFIG_I2C_INTERFACE)
        inst.write = _2SMPB_i2c_writes;
        inst.read = _2SMPB_i2c_reads;
        inst.wait_ms = _wait_ms;

    #elif defined(_2SMPB_WITH_MBED) && !defined(_2SMPB_CONFIG_I2C_INTERFACE)
        inst.write = _2SMPB_spi_writes;
        inst.read = _2SMPB_spi_reads;
        inst.wait_ms = _wait_ms;

        spi.format(8, 0);   // Flame=8bit,mode=0(POL=0,PHA=0)
        spi.frequency(1000000L);
        _2SMPB_spi_setss(0);

        // (optional) sample code to reset by GPIO
        #if 0
            gpio1 = 0;
            inst.wait_ms(1);
            gpio1 = 1;
            inst.wait_ms(1);
            gpio1 = 0;
            inst.wait_ms(10);
        #endif
    #endif

    // (optional) Reset by GPIO...
    #if defined(_2SMPB_WITH_MBED)
        gpio1 = 0;
        inst.wait_ms(1);
        gpio1 = 1;
        inst.wait_ms(1);
        gpio1 = 0;
        inst.wait_ms(10);
    #endif

    // initialize
    if (_2SMPB02_init(&inst, 5, 2, _2SMPB02_CTRL_MODE)) {
        exit(0);   // sensor is none.
    }
```

```
    for (int i = 0; i < 100; i++) {
        _2SMPB02_run(&inst);
        #if defined(_2SMPB_WITH_MBED)
            pc.printf("%10.1f,%6.3f¥n", inst.Po, inst.Tx / 256.0);
        #else
            printf("%10.1f,%6.3f¥n", inst.Po, inst.Tx / 256.0);
        #endif
        inst.wait_ms(800);
    }
}

#endif   // _2SMPB_MOCK_DEBUG
// end of file {{{1
// vi:ft=c:et:nowrap:fdm=marker
```

7.3. 2SMPB_02E.c

```
/** ********************************************************************************
 * ¥file 2SMPB_02E.c
 * ¥brief define 2SMPB-02 functions around sensor hardware behaviors
 *
 * ¥license {
 * This is free and unencumbered software released into the public domain.
 *
 * Anyone is free to copy, modify, publish, use, compile, sell, or distribute
 * this software, either in source code form or as a compiled binary,
 * for any purpose, commercial or non-commercial, and by any means.
 *
 * In jurisdictions that recognize copyright laws, the author or authors of this
 * software dedicate any and all copyright interest in the software to the
 * public domain. We make this dedication for the benefit of the public at large
 * and to the detriment of our heirs and successors. We intend this dedication
 * to be an overt act of relinquishment in perpetuity of all present and future
 * rights to this software under copyright law.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
 * AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
 * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 * }
 *
 * WARNING:
 * This is only for evaluation purpose and
 * Users must be follow the calculation in datasheet.
 *
 * $Rev: f918b7b $
 * $Id: 2017-05-29 16:18:54 +0900 $
```

```
 ******************************************************************************/
#include <stdint.h>
#include <stdbool.h>
#include <math.h>

#if defined(DEBUG)
	#include <stdio.h>
#endif

#define _2SMPB_PARAM_02E 1
#include "2SMPB02.h"


/** selected functions <!-- {{{1 --> */
	#define _2SMPB_CALC_CUBIC

	#define _2SMPB_setup _2SMPBC_setup
	#define _2SMPB_clear_coef _2SMPBC_clear_coef_dbl
	#define _2SMPB_convTx _2SMPBC_convTx_dbl
	#define _2SMPB_get_pressure _2SMPBC_get_pressure_dbl



/** local functions <!-- {{{1 --> */
static bool _2SMPB02_clear(_2SMPB02* obj);


/** buffer for sensor communications */
static uint8_t cmd[_2SMPB_PROM_DATA_LEN];


/** convu16_s16: helper: convert unsinged 16bit to signed <!-- {{{1 -->
 */
int16_t convu16_s16(uint16_t d) {
	int32_t ret;

	if ((d & 0x8000) != 0) {
		ret = (int32_t)d - 0x10000;
	} else {
		ret = d;
	}
	return ret;
}

/** convu20_s32: helper: convert unsinged 16bit to signed <!-- {{{1 -->
 */
int32_t convu20_s32(uint32_t d) {
	int32_t ret;

	if ((d & 0x80000) != 0) {
		ret = (int32_t)d - 0x100000;
```

```
	} else {
		ret = d;
	}
	return ret;
}

/** convu24_s32: helper: convert unsinged 16bit to signed <!-- {{{1 -->
 */
int32_t convu24_s32(uint32_t d) {
	int32_t ret;

	if ((d & 0x800000) != 0) {
		ret = (int32_t)d - 0x1000000;
	} else {
		ret = (int32_t)d;
	}
	return ret;
}

/** _2SMP02_init: initialize sensor data <!-- {{{1 -->
 *
 * **note** I2C speed must be normal or fast mode (<400kHz)
 *		during initializing the sensor.
 *
 * 1. initialize memory.
 * 2. software reset to the sensor.
 * 3. (optional) change I2C speed.
 * 4. find out the sensor on I2C/SPI.
 * 5. set IIR repeat to 16times.
 * 6. read calculate parameters via I2C/SPI.
*/
bool _2SMPB02_init(_2SMPB02* obj, uint8_t p, uint8_t t, uint8_t m) {
	// 1. initialize memory.
	_2SMPB02_clear(obj);
	obj->avg_p = p;		// oversampling of pressure
	obj->avg_t = t;		// oversampling of temperature
	obj->opmode = m;	// power mode: 'sleep', 'forced' or 'normal'

	// 2. software reset to the sensor.
	cmd[0] = _2SMPB02_RESET_CODE;
	obj->write(_2SMPB02_RESET_REG, cmd, 1);
	obj->wait_ms(10);	// 10ms

	#if defined(_2SMPB_CONFIG_I2C_INTERFACE) && defined(_2SMPB02_I2CHS)
		// 3. change I2C speed.
		cmd[0] = _2SMPB02_I2CHS_MASTERCODE;
		obj->write(_2SMPB02_I2CHS_REG, cmd, 1);

		// after this point, change your I2C Read/Write functions to
		// * send master code (0000 1??? <= ??? is _2SMPB02_I2CHS_MASTERCODE)
		// * change speed
```

```
        // * restore speed after stop condition.
    #endif

    // 4. find out the sensor by reading CHIPID & CHIPID2
    if (!_2SMPB02_find(obj)) {
        return true;
    }

    // 5. set IIR repeat to 16times.
    cmd[0] = _2SMPB02_IIR_16;
    obj->write(_2SMPB02_IIR_REG, cmd, 1);

    // 5. read calculation parameters SMPB02 calibparam structure
    _2SMPB02_get_cal_param(obj);

        // 7. start the sensor.
        _2SMPB02_set_ctrl_meas(obj);
    #endif

    return false;
}


/** _2SMPB02_clear: initialize sensor calculate parameters. <!-- {{{1 -->
 *
 */
bool _2SMPB02_clear(_2SMPB02* obj) {
    int i;

    for (i = 0; i < _2SMPB_PROM_DATA_LEN; i++) {
        obj->otps[i] = 0;
    }

    _2SMPB_clear_coef(obj);

    obj->chip_id1 = 0;

    obj->digitTemp = 0;
    obj->digitPres = 0;
    obj->Tx = 0.0;
    obj->Po = 0.0;
    obj->avg_p = 0;
    obj->avg_t = 0;
    obj->opmode = 0;

    return true;
}


/** _2SMPB02_find: find out the sensor <!-- {{{1 -->
 * by reading CHIP_ID_REG and CHIP_ID2_REG.
```

```
 */
bool _2SMPB02_find(_2SMPB02* obj) {
	int16_t err = 0;

	// read out Chip Id
	err += obj->read(_2SMPB02_CHIP_ID1_REG, cmd, 1);
	obj->chip_id1 = cmd[0];

	if (err != 0) {
		return false;
	}
	return obj->chip_id1 == _2SMPB02_CHIP_ID1;
}


/** _2SMPB02_get_cal_param: read all sensor calculate parameters <!-- {{{1 -->
 *
 */
int16_t _2SMPB02_get_cal_param(_2SMPB02* obj) {
	int16_t err = 0;

	err += obj->read(_2SMPB02_PROM_START_ADDR,
				obj->otps, _2SMPB_PROM_DATA_LEN);

	_2SMPB_setup(obj);
	return err;
}


/** _2SMPB02_set_ctrl_meas:  <!-- {{{1 -->
 *
 */
int16_t _2SMPB02_set_ctrl_meas(_2SMPB02* obj) {
	cmd[0] = (obj->avg_t << 5) | (obj->avg_p << 2) | obj->opmode;
	// oc_dbg("ctrl_meas: %x¥n", cmd[0]);
	return obj->write(_2SMPB02_CTRL_MEAS_REG, cmd, 1);
}


/** _2SMPB02_combine_txd: combine data stream to 24bit data <!-- {{{1 -->
 *
 */
int32_t _2SMPB02_combine_txd(uint8_t* data) {
	// "data" format
	// <--[0]-> <--[1]-> <--[2]->
	// <--msb-> <--lsb-> <-xlsb->
	// 76543210 76543210 76543210
	// dddddddd dddddddd dddddddd  d: valid_data, 0: unimplemented
	// <-----20bit---------->      average==1
	// <-----21bit---------->      average==2
	// <-----22bit---------->      average==3
```

```
	// <-----23bit-------------> average==4
	// <-----24bit--------------->average>=5
	uint32_t ud;
	int32_t ret;

	ud = (((uint32_t)data[0] << 16) |
		((uint32_t)data[1] << 8) |
		 (uint32_t)data[2]);
	ret = (int32_t)ud - 0x800000;
	// oc_dbg("combine_txd: %x = %d¥n", ret, ret);
	return ret;
}


/** _2SMPB02_get_temp_data: Get temperature data <!-- {{{1 -->
 *
 */
int32_t _2SMPB02_get_temp_data(_2SMPB02* obj) {
	int16_t err = 0;

	err += obj->read(_2SMPB02_TEMP_TXD2, cmd, 3);
	if (err != 0) {return 0;}

	return _2SMPB02_combine_txd(cmd);
}


/** _2SMPB02_get_press_data: Get pressure digit <!-- {{{1 -->
 *
 */
int32_t _2SMPB02_get_press_data(_2SMPB02* obj) {
	int32_t err = 0;

	err += obj->read(_2SMPB02_PRES_TXD2, cmd, 3);
	if (err != 0) {return 0;}

	return _2SMPB02_combine_txd(cmd);
}


/** _2SMPB02_is_running: Get sensor status <!-- {{{1 -->
 *
 */
bool _2SMPB02_is_running(_2SMPB02* obj) {
	int32_t err = 0;

	err += obj->read(_2SMPB02_STATUS_REG, cmd, 1);
	if (err != 0) {return true;}

	if (cmd[0] & (_2SMPB02_STATUS_MEAS | _2SMPB02_STATUS_OTP)) {
	    return true;
```

```
	}
	return false;
}


/** _2SMPB02_run: The total flow to get pressure data in [Pa] <!-- {{{1 -->
 *
 */
int16_t _2SMPB02_run(_2SMPB02* obj) {
	if (obj->coef == NULL) {
		obj->digitTemp = 0;
		obj->digitPres = 0;
		obj->Tx = 0;
		obj->Po = 0;
		return _2SMPB02_NOT_INITIALIZED;
	}
	#endif

	obj->digitTemp = _2SMPB02_get_temp_data(obj);
	obj->digitPres = _2SMPB02_get_press_data(obj);

	obj->Tx = _2SMPB_convTx(obj, obj->digitTemp);
	obj->Po = _2SMPB_get_pressure(obj, obj->digitPres, obj->Tx);

	#if !defined(_2SMPB02_NORMALMODE)
		_2SMPB02_set_ctrl_meas(obj);
	#endif
	return _2SMPB02_OK;
}


/** compensation parameter of the 2SMPB series Quadratic compensation,
*/
typedef struct tag_2SMPBC_COEF {
	/** initialized calculation parameters */
	bool is_initialized;

	/** otp data by multi bytes */
	uint32_t B00, A0_;		// 20Q4
	uint16_t BT1, BP1;
	uint16_t BT2, B11, BP2;
	uint16_t B12, B21, BP3;
	uint16_t A1_, A2_;

	/** otp data by multi bytes */
	double b00, bt1, bt2, bp1, b11, bp2, b12, b21, bp3;
	double a0_, a1_, a2_;
} _2SMPBC_COEF;

/** need to change multi instances for multi sensors */
static _2SMPBC_COEF _2smpbc_coef = {
```

```
	.is_initialized = false
};

/** compensation parameter values: _A_ is center, _S_ is range. */
#define _2SMPB_A_BT1 (_2SMPBCUBIC_CA_BT1)
#define _2SMPB_A_BT2 (_2SMPBCUBIC_CA_BT2)
#define _2SMPB_A_BP1 (_2SMPBCUBIC_CA_BP1)
#define _2SMPB_A_B11 (_2SMPBCUBIC_CA_B11)
#define _2SMPB_A_BP2 (_2SMPBCUBIC_CA_BP2)
#define _2SMPB_A_B12 (_2SMPBCUBIC_CA_B12)
#define _2SMPB_A_B21 (_2SMPBCUBIC_CA_B21)
#define _2SMPB_A_BP3 (_2SMPBCUBIC_CA_BP3)
#define _2SMPB_A_A1_ (_2SMPBCUBIC_CA_A1_)
#define _2SMPB_A_A2_ (_2SMPBCUBIC_CA_A2_)

#define _2SMPB_S_BT1 ((double)(1.0 / _2SMPBCUBIC_CS_BT1 * 32767))
#define _2SMPB_S_BT2 ((double)(1.0 / _2SMPBCUBIC_CS_BT2 * 32767))
#define _2SMPB_S_BP1 ((double)(1.0 / _2SMPBCUBIC_CS_BP1 * 32767))
#define _2SMPB_S_B11 ((double)(1.0 / _2SMPBCUBIC_CS_B11 * 32767))
#define _2SMPB_S_BP2 ((double)(1.0 / _2SMPBCUBIC_CS_BP2 * 32767))
#define _2SMPB_S_B12 ((double)(1.0 / _2SMPBCUBIC_CS_B12 * 32767))
#define _2SMPB_S_B21 ((double)(1.0 / _2SMPBCUBIC_CS_B21 * 32767))
#define _2SMPB_S_BP3 ((double)(1.0 / _2SMPBCUBIC_CS_BP3 * 32767))
#define _2SMPB_S_A1_ ((double)(1.0 / _2SMPBCUBIC_CS_A1_ * 32767))
#define _2SMPB_S_A2_ ((double)(1.0 / _2SMPBCUBIC_CS_A2_ * 32767))

#define _2SMPBCUBIC_OFFSET_B00 0
#define _2SMPBCUBIC_OFFSET_BT1 2
#define _2SMPBCUBIC_OFFSET_BT2 4
#define _2SMPBCUBIC_OFFSET_BP1 6
#define _2SMPBCUBIC_OFFSET_B11 8
#define _2SMPBCUBIC_OFFSET_BP2 10
#define _2SMPBCUBIC_OFFSET_B12 12
#define _2SMPBCUBIC_OFFSET_B21 14
#define _2SMPBCUBIC_OFFSET_BP3 16
#define _2SMPBCUBIC_OFFSET_A0_ 18
#define _2SMPBCUBIC_OFFSET_A1_ 20
#define _2SMPBCUBIC_OFFSET_A2_ 22
#define _2SMPBCUBIC_OFFSET_B00_A0_EX   24


/** local funtions */
static void _otp2coef_dbl(_2SMPB02* obj);


/** <!-- _2SMPBC_clear_coef_dbl {{{1 --> combine 8 digits to word variables
 */
bool _2SMPBC_clear_coef_dbl(_2SMPB02* obj) {
	_2SMPBC_COEF* coe;
	obj->coef = (void*)(coe = &_2smpbc_coef);
```

```
	coe->B00 = 0;
	coe->BT1 = 0;
	coe->BT2 = 0;
	coe->BP1 = 0;
	coe->B11 = 0;
	coe->BP2 = 0;
	coe->A0_ = 0;
	coe->A1_ = 0;
	coe->A2_ = 0;

	coe->b00 = 0.0;
	coe->bt1 = 0.0;
	coe->bt2 = 0.0;
	coe->bp1 = 0.0;
	coe->b11 = 0.0;
	coe->bp2 = 0.0;
	coe->a0_ = 0.0;
	coe->a1_ = 0.0;
	coe->a2_ = 0.0;

	coe->is_initialized = false;
	return false;
}



/** _joinotp: combine 8 digits to word variables <!-- {{{1 -->
 *
 */
static void _joinotp(_2SMPB02* obj) {
	_2SMPBC_COEF* coe = (_2SMPBC_COEF*)obj->coef;
	uint8_t* buf = obj->otps;
	uint8_t ex;

	coe->B00 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_B00);
	coe->BT1 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_BT1);
	coe->BT2 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_BT2);
	coe->BP1 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_BP1);
	coe->B11 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_B11);
	coe->BP2 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_BP2);
	coe->B12 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_B12);
	coe->B21 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_B21);
	coe->BP3 = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_BP3);
	coe->A0_ = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_A0_);
	coe->A1_ = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_A1_);
	coe->A2_ = _2SMPB_MACRO_BYTES_U16(buf, _2SMPBCUBIC_OFFSET_A2_);

	// join extra 4bits.
	ex = buf[_2SMPBCUBIC_OFFSET_B00_A0_EX];
	coe->A0_ = (coe->A0_ << 4) | (ex & 0x0F);		// 20Q4
	coe->B00 = (coe->B00 << 4) | ((ex >> 4) & 0x0F);	// 20Q4
```

```
	oc_dbg("B00: %x(uint)¥n", coe->B00);
	oc_dbg("BT1: %x(uint)¥n", coe->BT1);
	oc_dbg("BT2: %x(uint)¥n", coe->BT2);
	oc_dbg("BP1: %x(uint)¥n", coe->BP1);
	oc_dbg("B11: %x(uint)¥n", coe->B11);
	oc_dbg("BP2: %x(uint)¥n", coe->BP2);
	oc_dbg("B12: %x(uint)¥n", coe->B12);
	oc_dbg("B21: %x(uint)¥n", coe->B21);
	oc_dbg("BP3: %x(uint)¥n", coe->BP3);
	oc_dbg("A0_: %x(uint), %d¥n", coe->A0_, coe->A0_);
	oc_dbg("A1_: %x(uint)¥n", coe->A1_);
	oc_dbg("A2_: %x(uint)¥n", coe->A2_);

	coe->is_initialized = true;
}



/** _2SMPBC_setup: calcurate parameters from I2C data stream <!-- {{{1 -->
 *
 */
void _2SMPBC_setup(_2SMPB02* obj) {
	/** TODO: multiple DUTs */
	obj->coef = (void*)&_2smpbc_coef;
	_joinotp(obj);
	_otp2coef_dbl(obj);
}



/** _otp2coef_dbl: convert 16bit words to float parameters <!-- {{{1 -->
 *
 */
static void _otp2coef_dbl(_2SMPB02* obj) {
	_2SMPBC_COEF* coe = (_2SMPBC_COEF*)obj->coef;

	if (!coe->is_initialized) {
		/** (coe->B00 or etc) is not valid in this time. */
		return;
	}

	coe->b00 = (double)convu20_s32(coe->B00) / 16.0;	// 20Q4 => 1.0f
	coe->bt1 = (double)convu16_s16(coe->BT1) / _2SMPB_S_BT1 + _2SMPB_A_BT1;
	coe->bt2 = (double)convu16_s16(coe->BT2) / _2SMPB_S_BT2 + _2SMPB_A_BT2;

	coe->bp1 = (double)convu16_s16(coe->BP1) / _2SMPB_S_BP1 + _2SMPB_A_BP1;
	coe->b11 = (double)convu16_s16(coe->B11) / _2SMPB_S_B11 + _2SMPB_A_B11;
	coe->bp2 = (double)convu16_s16(coe->BP2) / _2SMPB_S_BP2 + _2SMPB_A_BP2;

	coe->b12 = (double)convu16_s16(coe->B12) / _2SMPB_S_B12 + _2SMPB_A_B12;
```

```
	coe->b21 = (double)convu16_s16(coe->B21) / _2SMPB_S_B21 + _2SMPB_A_B21;
	coe->bp3 = (double)convu16_s16(coe->BP3) / _2SMPB_S_BP3 + _2SMPB_A_BP3;

	coe->a0_ = (double)convu20_s32(coe->A0_) / 16.0;   // 20Q4 => 1.0f
	coe->a1_ = (double)convu16_s16(coe->A1_) / _2SMPB_S_A1_ + _2SMPB_A_A1_;
	coe->a2_ = (double)convu16_s16(coe->A2_) / _2SMPB_S_A2_ + _2SMPB_A_A2_;

	oc_dbg("b00,bt1,bt2 : %6.3e, %6.3e, %6.3e¥n",
		coe->b00, coe->bt1, coe->bt2);
	oc_dbg("bp1,b11,bp2 : %6.3e, %6.3e, %6.3e¥n",
		coe->bp1, coe->b11, coe->bp2);
	oc_dbg("b12,b21,bp3 : %6.3e, %6.3e, %6.3e¥n",
		coe->b12, coe->b21, coe->bp3);
	oc_dbg("a0 ,a1 ,a2   : %6.3e, %6.3e, %6.3e¥n",
		coe->a0_, coe->a1_, coe->a2_);
}

/** _2SMPBC_convTx_dbl: convert temperature digit to 256 degC <!-- {{{1 -->
 *
 */
double _2SMPBC_convTx_dbl(_2SMPB02* obj, int32_t dt) {
	double tx;
	_2SMPBC_COEF* coe = (_2SMPBC_COEF*)obj->coef;
	if (!coe || !coe->is_initialized) {return 0.0;}

	tx = coe->a0_ + coe->a1_ * dt + coe->a2_ * dt * dt;
	#if defined(_2SMPB_MOCK_DEBUG)
		printf("calc : Tx: %x -> %f (%f,%f,%f)¥n",
			dt, tx / 256.0, coe->a0_, coe->a1_ * dt, coe->a2_ * dt * dt);
	#endif
	return tx;
}


/** _2SMPBC_get_pressure_dbl: convert digit data to pressure[Pa] <!-- {{{1 -->
 *
 */
double _2SMPBC_get_pressure_dbl(_2SMPB02* obj, int32_t dp, double tx) {
	double po, p0x, p1x, p2x, p3x;
	_2SMPBC_COEF* coe = (_2SMPBC_COEF*)obj->coef;
	if (!coe || !coe->is_initialized) {return 0.0;}

	p0x = coe->b00 + coe->bt1 * tx + coe->bt2 * tx * tx;
	p1x = (coe->bp1 + coe->b11 * tx) * dp;
	p2x = coe->bp2 * dp * dp;
	p3x = coe->bp3 * dp * dp * dp
		+ coe->b12 * dp * tx * tx
		+ coe->b21 * dp * dp * tx;
	po = p0x + p1x + p2x + p3x;
	return po;
}
```

```
#if !defined(_2SMPB_MOCK_DEBUG)

static _2SMPB02 inst;   // Sensor struct

#if defined(_2SMPB_WITH_MBED) && defined(_2SMPB_CONFIG_I2C_INTERFACE)
    #include "mbed.h"

    I2C i2c(p28, p27);

    Serial pc(USBTX, USBRX);
    DigitalOut gpio1(p20);

    static uint8_t mbed_cmd[_2SMPB_PROM_DATA_LEN];

    // low level function definition
    int _2SMPB_i2c_write(uint8_t addr, uint8_t* buf, int n) {
        return i2c.write(addr, (const char*)buf, n);
    }

    int _2SMPB_i2c_read(uint8_t addr, uint8_t* buf, int n) {
        return i2c.read(addr, (char*)buf, n);
    }

    int _2SMPB_i2c_writes(uint8_t reg, uint8_t* buf, int n) {
        int i, err = 0;

        mbed_cmd[0] = reg;
        for (i = 0; i < n; i++) {
            mbed_cmd[1 + i] = buf[i];
        }
        err += _2SMPB_i2c_write(_2SMPB02_I2C_ADDR_WR, mbed_cmd, n + 1);
        return err;
    }

    int _2SMPB_i2c_reads(uint8_t reg, uint8_t* buf, int n) {
        int err = 0;

        mbed_cmd[0] = reg;
        err += _2SMPB_i2c_write(_2SMPB02_I2C_ADDR_WR, mbed_cmd, 1);
        err += _2SMPB_i2c_read(_2SMPB02_I2C_ADDR_RD, buf, n);
        return err;
    }

#elif defined(_2SMPB_WITH_MBED) && !defined(_2SMPB_CONFIG_I2C_INTERFACE)
    #include "mbed.h"

    #if defined(_2SMPB_WITH_MBED_SPI1)
        SPI spi(p5, p6, p7);
```

```
        DigitalOut gpio_ss(p8);
    #elif defined(_2SMPB_WITH_MBED_SPI2)
        SPI spi(p11, p12, p13);
        DigitalOut gpio_ss(p14);
    #else
        #error please set 1 or 2 to _2SMPB_WITH_MBED for select peripheral.
    #endif

    Serial pc(USBTX, USBRX);
    DigitalOut gpio1(p20);

    void _2SMPB_spi_setss(bool hi_low) {
        // hi_low: 1 => hi, 0 => low.
        gpio_ss = hi_low;
    }

    uint8_t _2SMPB_spi_write(uint8_t val) {
        return spi.write(val);
    }

    int _2SMPB_spi_writes(uint8_t reg, uint8_t* buf, int n) {
        int i;

        _2SMPB_spi_setss(0);
        _2SMPB_spi_write(reg & 0x7F);
        for (i = 0; i < n; i++) {
            _2SMPB_spi_write(buf[i]);
        }
        wait_ms(1);
        _2SMPB_spi_setss(1);
        return 0;
    }

    int _2SMPB_spi_reads(uint8_t reg, uint8_t* buf, int n) {
        int i;

        _2SMPB_spi_setss(0);
        _2SMPB_spi_write(reg);
        for (i = 0; i < n; i++) {
            buf[i] = _2SMPB_spi_write(0xff);
        }
        wait_ms(1);
        _2SMPB_spi_setss(1);
        return 0;
    }

#else
    void exit();      // exit() is a mbed function.
#endif

#if defined(_2SMPB_WITH_MBED)
```

```
    void _wait_ms(uint32_t usec) {
        wait_us((int)usec * 1000);
    }
#endif


int main() {
    // select functions by your system
    #if 0
        inst.read = your_read_function;
        inst.write = your_write_function;
        inst.wait_ms = your_wait_micro_sec_function;

    #elif defined(_2SMPB_WITH_MBED) && defined(_2SMPB_CONFIG_I2C_INTERFACE)
        inst.write = _2SMPB_i2c_writes;
        inst.read = _2SMPB_i2c_reads;
        inst.wait_ms = _wait_ms;

    #elif defined(_2SMPB_WITH_MBED) && !defined(_2SMPB_CONFIG_I2C_INTERFACE)
        inst.write = _2SMPB_spi_writes;
        inst.read = _2SMPB_spi_reads;
        inst.wait_ms = _wait_ms;

        spi.format(8, 0);   // Flame=8bit,mode=0(POL=0,PHA=0)
        spi.frequency(1000000L);
        _2SMPB_spi_setss(0);

        // (optional) sample code to reset by GPIO
        #if 0
            gpio1 = 0;
            inst.wait_ms(1);
            gpio1 = 1;
            inst.wait_ms(1);
            gpio1 = 0;
            inst.wait_ms(10);
        #endif
    #endif

    // (optional) Reset by GPIO...
    #if defined(_2SMPB_WITH_MBED)
        gpio1 = 0;
        inst.wait_ms(1);
        gpio1 = 1;
        inst.wait_ms(1);
        gpio1 = 0;
        inst.wait_ms(10);
    #endif

    // initialize
    if (_2SMPB02_init(&inst, 5, 2, _2SMPB02_CTRL_MODE)) {
        exit(0);   // sensor is none.
```

```
    }

    for (int i = 0; i < 100; i++) {
        _2SMPB02_run(&inst);
        #if defined(_2SMPB_WITH_MBED)
            pc.printf("%10.1f,%6.3f¥n", inst.Po, inst.Tx / 256.0);
        #else
            printf("%10.1f,%6.3f¥n", inst.Po, inst.Tx / 256.0);
        #endif
        inst.wait_ms(800);
    }
}

#endif   // _2SMPB_MOCK_DEBUG
// end of file {{{1
// vi:ft=c:et:nowrap:fdm=marker
```

8. Warranty and Limited Liability

Thank you for your usage of products of Omron Corporation (“Omron”). Without any special agreements, these terms and conditions shall apply to all transactions regardless of who sells. Place an order, accepting these terms and conditions.

8.1. Definitions

The following terms used herein have following meaning.

(1) Omron Products; Electronic components sold by Omron

(2) Catalogues; Any and all catalogues (including the Components Catalogue), specifications, instructions and manuals relating to Omron Products, including electronically provided data.

(3) Conditions; Use conditions, rating, performance, operating environment, handling procedure, precautions and/or prohibited use of Omron Products described in the Catalogues.

(4) User Application(s); Application of Omron Products by a customer, including but not limited to embedding Omron Products into customer’s components, electronic circuit boards, devices, equipments or systems

(5) Fitness; (a)performance, (b) no infringement of intellectual property of third party, (c) compliance with laws and regulations and (d)conformity to various standards by Omron Products in User Applications.

8.2. Note about Descriptions

Please understand following as to contents of the Catalogues.

(1) Rating and performance is tested separately. Combined conditions are not warranted.

(2) Reference data is intended to be used just for reference. Omron does NOT warrant that the Omron Product can work properly in the range of reference data.

(3) Examples are intended for reference. Omron does not warrant the Fitness in usage of the examples.

(4) Omron may discontinue Omron Products or change specifications of them because of improvements or other reasons.

8.3. Note about Use

Please understand followings as to your adoption and use of Omron Products

(1) Please use the product in conformance to the Conditions, including rating and performance.

(2) Please confirm the Fitness and decide whether or not Omron Products are able to be adopted in the User Application.

(3) Omron will not warrant any items in 1.(5) (a) to (d) of User Application nor the Fitness.

(4) If you use Omron Products in the application below, please ensure followings; (i) allowance in aspect of rating and performance, (ii) safety design which can minimize danger of the Application when the product does not work properly and (iii) periodical maintenance of the product and the Application.

(a) Applications requiring safety, including, without limitation, nuclear control facilities, combustion facilities, aerospace and aviation facilities, railroad facilities, elevating facilities, amusement facilities, medical facilities, safety devices or other applications which has possibility to influence lives or bodies

(b) Applications requiring high reliability, including, without limitation, supplying systems of gas, water and electric power and applications handling right, title, ownership or property, such as payment systems

(c) Applications in a harsh condition or environment, including, without limitation, outdoor facilities, facilities with potential of chemical contamination or electromagnetic interference, facilities with vibration or impact and facilities on continual operation for a long period

(d) Applications under conditions or environment which are not described in this specification

(5) Omron Products shown in this catalogue are not intended to be used in automotive applications (including two wheel vehicles). Please DO NOT use the Omron Products in the automotive application.

(6) The products contained in this catalog are not safety rated. They are not designed or rated for ensuring safety of persons, and should not be relied upon as a safety component or protective device for such purposes. Please refer to separate catalogs for Omron's safety rated products.

8.4. Warranty

Warranty of Omron Products is subject to followings.

(1) Warranty Period: One year after your purchase

(2) Coverage: Omron will provide free replacement of the malfunctioning Omron products with the same number of replacement/alternative products

(3) Exceptions: This warranty does not cover malfunctions caused by any of the following.

(a) Usage in the manner other than its original purpose

(b) Usage out of the Conditions

(c) Cause which could not be foreseen by the level of science and technology at the time of shipment of the product

(d) Causes outside Omron or Omron Products, including force majeure such as disasters

8.5. Limitation of Liability

The warranty described in these terms and conditions are a whole and sole liability for Omron products. There are no other warranties, expressed or implied. Omron and Distributors are not liable for any damages arisen from or relating to Omron products.

8.6. Programmable Products

Omron shall not be responsible for the user's programming of a programmable product, or any consequence thereof.

8.7. Export Controls

Customers of Omron products shall comply with all applicable laws and regulations of other relevant countries with regard to security export control, when exporting Omron products and/or technical documents or providing such products and/or documents to a non-resident.

EC200E

9. Contact

- **Omron Electronic Components Web**
  https://www.components.omron.com/
- **Contact Us**
  For further inquiry such as delivery, price, sample and/or specification, please contact your local agency or Omron sales representative.
  - **Global Sales Office**

  https://www.components.omron.com/web/en/ecb
  - **Mail Contact**

  https://www.components.omron.com/contact-us
  - **Phone**

  Business Management Division H.Q. Tel: **(81) 75-344-7146**
  Shiokoji Horikawa, Shimogyo-Ku, Kyoto, 600-8530 JAPAN

  Place an order, accepting this Terms and Conditions.
  https://www.components.omron.com/web/en/terms-of-use

10. History

Revision	DATE	Note
Rev 1.0	May 2018	New Released

Please check each region's Terms & Conditions by region website.

OMRON Corporation

Electronic and Mechanical Components Company

Regional Contact

Americas
https://www.components.omron.com/

Europe
http://components.omron.eu/

Asia-Pacific
https://ecb.omron.com.sg/

China
https://www.ecb.omron.com.cn/

Korea
https://www.omron-ecb.co.kr/

Japan
https://www.omron.co.jp/ecb/

Cat. No. A277-E1-01
0518 (0518)(O)