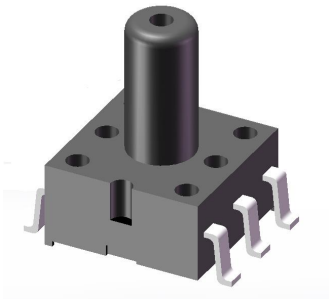
XGZP6859D PRESSURE SENSOR

FEATURES

- Wide Ranges: -100kPa···0kPa···1000kPa(show in Pressure Range Example)
- 3.0V ~ 5.5V Power Supply
- Gauge Pressure Type(Positive&Vacuum)
- For Non-corrosive Gas or Air
- Calibrated Digital Signal(I2C Interface)(Refer to XGZP6859A for Analog signal)
- Temp. Compensated: 0℃ ~ +60℃(32℉ ~ +140℉)
- Temperature Measurable
- Affordable Cost, Easy-to-use



√ RoHS

APPLICATIONS

- Medical&Healthcare
- Industrial&Automation
- Domestic Appliance
- Consumer Electronic
- Automotive Electronic

INTRODUCTION

XGZP6859D is a prefect silicon pressure sensor offering a ratiometric digital data(I2C interface) for reading relative pressure over the specified full scale pressure span.

The XGZP6859D incorporates a silicon piezoresistive pressure sensor chip and an interior signal-conditional Application Specific Integrated Circuit(ASIC) in a SOP6 package, which can be mounted directly on a standard PCB by SMT.

The XGZP6859D is fully calibrated and temperature compensated for specified span, so XGZP6859D pressure sensor satisfy the perfect accuracy, which is designed for a wide range of application in medical care&health, home appliances, consumer electronic, industry, automotive, IoT and other pneumatic devices etc by utilizing a microcontroller or microprocessor with I2C interface.

XGZP6859D pressure sensor is for high volume application at an affordable cost and perfect performance. Customized calibration parameter (e.g.pressure range etc.) are available.

PERFORMANCE SPECIFICATION

Unless otherwise specified, measurements were taken by 3.3Vdc with temperature of 25±1℃ and humidity from 25% ~ 85 % RH.

CHARACTERISTIC		MIN.	TYP.	MAX	UNIT
Available Pressure Range①		-100…-1 ~ 0 ~ 1…1000			kPa
Power Supply②		3.0	3.3	5.5	Vdc
Current Consumption	Operating Current	-	1.8	-	mA
	Standby Current	-	100	-	nA
Output Resolution③	Pressure	21			Bit
	Temperature	16			Bit
Total Accuracy	10kPa < Pressure ≤200kPa	-	-	±2	%FSS
	Pressure ≤10kPa or > 200kPa	-	-	±2.5	%FSS
Temperature Accuracy		-2		2	℃
Long Term Stability(1000 hr, 25°C)		-	-	±0.5	%FSS
Over Pressure④	Pressure ≤5kPa	-	5X	-	FSS
	5kPa < Pressure ≤200kPa	-	2.5X	-	FSS
	200kPa < Pressure	-	-	1.5X	FSS
Burst Pressure④	Pressure ≤5kPa	-	10X	-	FSS
	5kPa < Pressure ≤200kPa	-	3X	-	FSS
	200kPa < Pressure	-	-	2X	FSS
Compensation Temperature		0	-	60	℃
Operating Temperature		-30	-	105	℃
Storage Temperature		-40	-	125	℃
ESD Protection(Human Body Mode)		-	±2000	-	V
Response Time(Normal mode)		-	5	-	mS

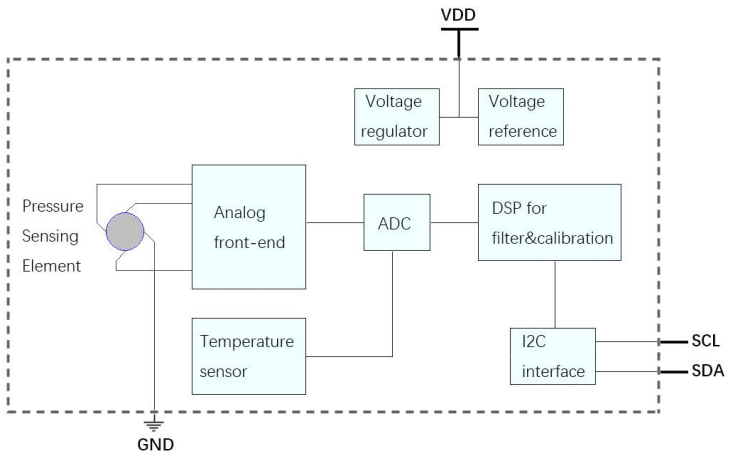
① The range cover all pressure ranges as shown as “PRESSURE RANGE EXAMPLE” list.

② Overload voltage(6.5Vdc above) or current(5mA above) may burn the IC and cause the sensor failure thoroughly.

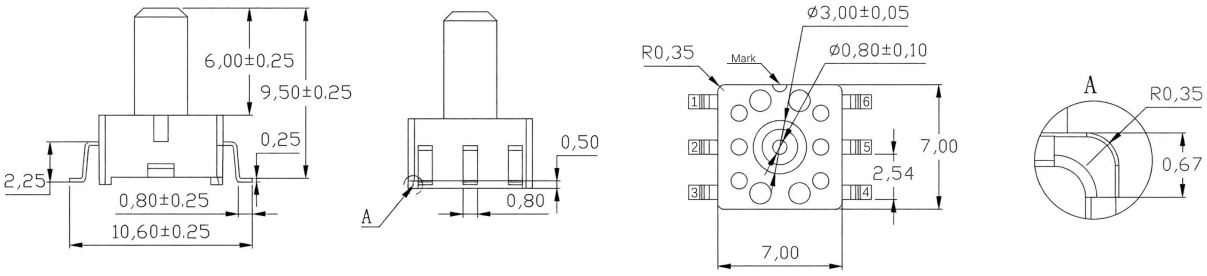
③ The highest data bit as the signed number.

④ The indicated value is general value, contact CFSensor for more information on specific pressure range.

BLOCK DIAGRAM



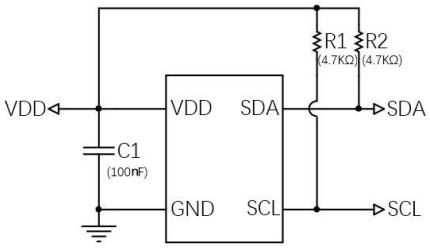
DIMENSION (Unit:mm Unspecified Tolerances:±0.2mm)



PIN DEFINITION

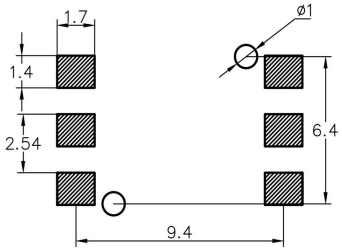
PIN1	PIN2	PIN3	PIN4	PIN5	PIN6
NC	VDD	NC	SDA	SCL	GND

SCL	Clock signal
NC	Do not connect to external circuitry or ground
GND	Ground
VDD	Voltage supply
SDA	Data signal(Send& Receive)



Recommended Application Circuit

FOOTPRINT(Unit:mm)

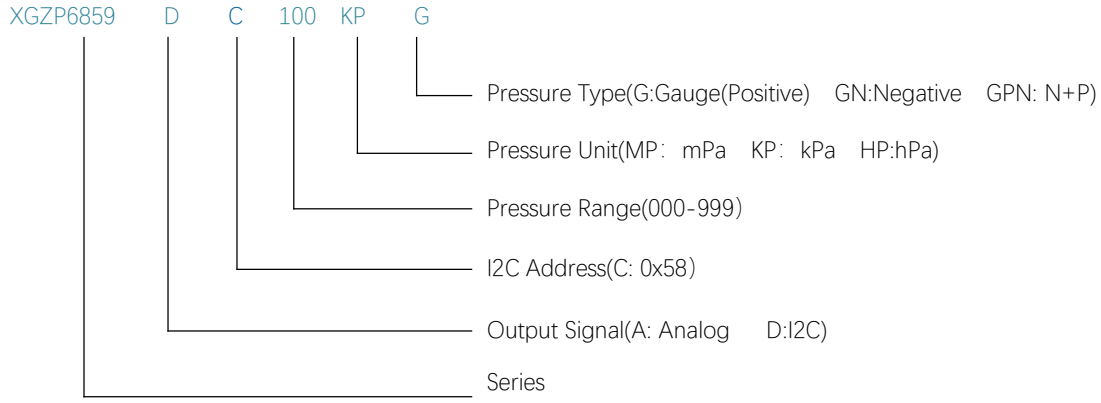


NOTE: FOOTPRINT LAYOUT FOR REFERENCE ONLY

Symbol FootPrint 3D

CONTACT CFSENSOR FOR ABOVE FILE IF REQUIRED

ORDER GUIDE



Note: Custom requirement or parameter(e.g pressure range, output etc,.), consult with CFSensor on Part Number.

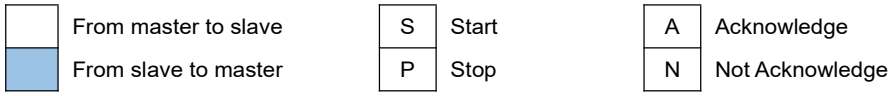
PRESSURE RANGE EXAMPLE

Notes: 1. Unit conversion: 1000hPa==100kPa=0.1MPa=1000mbar=1bar≈750mmHg≈14.5PSI≈10mH$_2$O;
2. Available for more custom pressure range e.g. -7 ~ 7kPa, , 0-3.92kPa etc,.

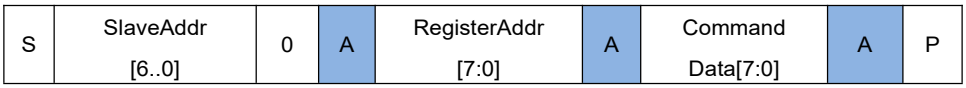
Pressure Range (kPa)	Pressure Range (by other units)	Part Number
0 ~ 2.5	0 ~ 25mbar / 0 ~ 250mmH$_2$O	XGZP6859DC025HPG
0 ~ 5	0 ~ 50mbar / 0 ~ 500mmH$_2$O	XGZP6859DC005KPG
0 ~ 10	0 ~ 100mbar / 0 ~ 75mmHg	XGZP6859DC010KPG
0 ~ 20	0 ~ 200mbar / 0 ~ 150mmHg	XGZP6859DC020KPG
0 ~ 40	0 ~ 400mbar / 0 ~ 300mmHg	XGZP6859DC040KPG
0 ~ 100	0 ~ 1bar /0 ~ 14.5PSI	XGZP6859DC100KPG
0 ~ 200	0 ~ 2bar / 0 ~ 29PSI	XGZP6859DC200KPG
0 ~ 500	0 ~ 5bar / 0 ~ 72.5PSI	XGZP6859DC500KPG
0 ~ 700	0 ~ 7bar / 0 ~ 100PSI	XGZP6859DC700KPG
0 ~ 1000	0 ~ 10bar / 0 ~ 29PSI / 0 ~ 1MPa	XGZP6859DC001MPG
-100 ~ 0	-1 ~ 0bar / -14.5 ~ 0PSI	XGZP6859DC100KPGN
-30 ~ 0	-300 ~ 0mbar / -4.35 ~ 0PSI	XGZP6859DC030KPGN
-20 ~ 0	-200 ~ 0mbar / -2.9 ~ 0PSI	XGZP6859DC020KPGN
-5 ~ 5	-50 ~ 50mbar / -500 ~ 500mmH$_2$O	XGZP6859DC005KPGPN
-40 ~ 40	-400 ~ 400mbar / -300 ~ 300mmHg	XGZP6859DC040KPGPN
-100 ~ 100	-1 ~ 1bar / -14.5 ~ 14.5PSI	XGZP6859DC100KPGPN
-100 ~ 300	-1 ~ 3bar / -14.5 ~ 43.5PSI	XGZP6859DC300KPGPN
-100 ~ 700	-1 ~ 7bar / -14.5 ~ 100PSI	XGZP6859DC700KPGPN
-100 ~ 1000	-1 ~ 10bar / -14.5 ~ 145PSI / 0.1 ~ 1MPa	XGZP6859DC001MPGPN
★Above P/N is example only, consult CFSensor whether required pressure range is under normal production before place order.		

I2C INTERFACE

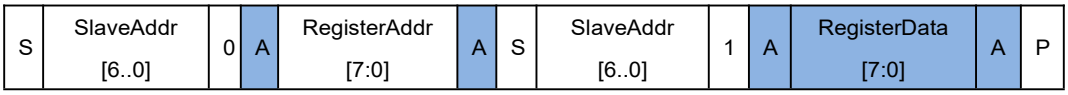
The I2C bus uses SCL and SDA as signal lines, both of which are connected to VDD through pull-up resistors (typ.value: 4.7K) and remain high level when not communicating. I2C device factory setting slave address: 0X58



Write Operation



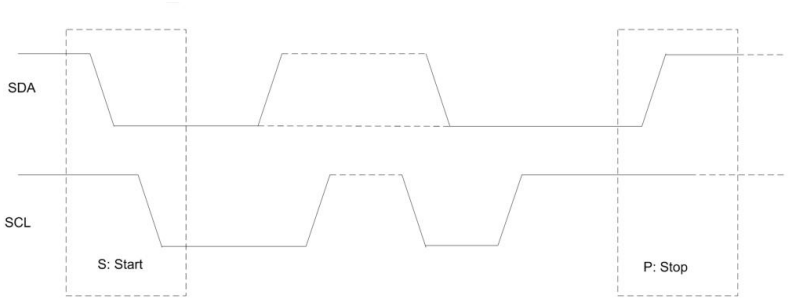
Read Operation



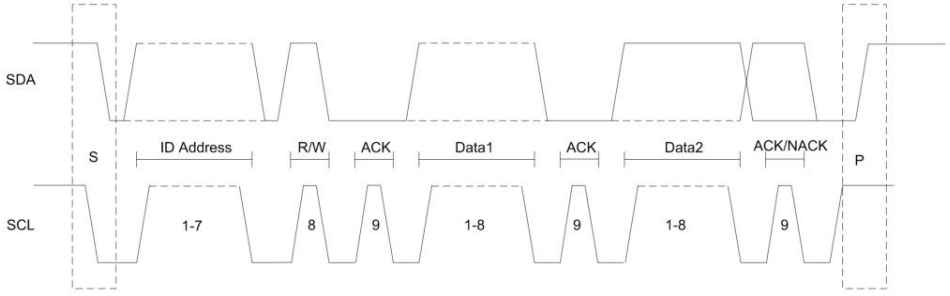
I2C TIME DIAGRAM

The I2C interface protocol has special bus signal conditions. Start (S), stop (P) and binary data conditions are shown below. At start condition, SCL is high and SDA has a falling edge. Then the slave address is sent. After the 7 address bits, the direction control bit R/W selects the read or write operation. When a slave device recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle.

At stop condition, SCL is also high, but SDA has a rising edge. Data must be held stable at SDA when SCL is high. Data can change value at SDA only when SCL is low.



I2C PROTOCAL



GENERAL REGISTER DESC.

Add.	Desc.	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0x00	ID	R	1	0	1	1	0	0	0	0/1
0x01	Chip_Control	R/W	/					ONE_PT[2]	/	Active[0]
0x02	CFG_OSR	R/W	OSR_T[7:5]			OSR_P[4:2]			MODE[1:0]	
0x03	CFG_MEAS	R/W	/		T_SB[5:3]			PT_R[2:0]		
0x04	P_data	R	Pressure Data out High byte[7:0]							
0x05	P_data	R	Pressure Data out Middle byte[7:0]							
0x06	P_data	R	Pressure Data out Least byte[7:0]							
0x07	T_data	R	Temperature Data out High byte[7:0]							
0x08	T_data	R	Temperature Data out Least byte[7:0]							
0x20-21	Cal_coff	R/W	Temperature Calibration Parameters [7:0]							

Reg0x00: I2C device address, the default address is 0x58H.

Reg0x01: Chip activation control register

Reg0x02: CFG_OSR(Oversampling control register)

Reg0x03: CFG_MEAS (Measurement command register)

Reg0x04-Reg0x06: Pressure data register

Reg0x07-Reg0x08: Temperature data register

Bit #	Name	Description	
0x01.[0]	ACTIVE	0b: Chip power down(Idle mode) 1b:Chip active	
0x01.[2]	One_PT	0b:Pressure measurement 1b:Temperature measurement	
0x02.[1:0]	MODE[1:0]	00b:Sleep mode 01b:Normal mode 10b:One shot mode	
0x02.[4:2]	OSR_P[2:0]	Oversampling rate of pressure measurement	
		000b: over sampling x 256 001b: over sampling x 512 010b: over sampling x 1024 011b: over sampling x 2048	100b: over sampling x 4096 101b: over sampling x 8192 110b: over sampling x 16384 111b: over sampling x 32768
0x02.[7:5]	OSR_T[2:0]	Oversampling rate of temperature measurement	
		000b: over sampling x 256 001b: over sampling x 512 010b: over sampling x 1024 011b: over sampling x 2048	100b: over sampling x 4096 101b: over sampling x 8192 110b: over sampling x 16384 111b: over sampling x 32768
0x03.[2:0]	PT_R[2:0]	Pressure/Temperature measurement ratio in normal mode	
		000b: 64/1 001b: 32/1 010b: 16/1 011b: 8/1 100b: 4/1 101b: 1/1	
0x03.[5:3]	T_SB[2:0]	Standby period setting in normal mode	
		000b: 0ms 001b: 62.5ms 010b: 125ms 011b: 250ms 100b: 500ms 101b: 750ms 110b: 1000ms 111b: 2000ms	

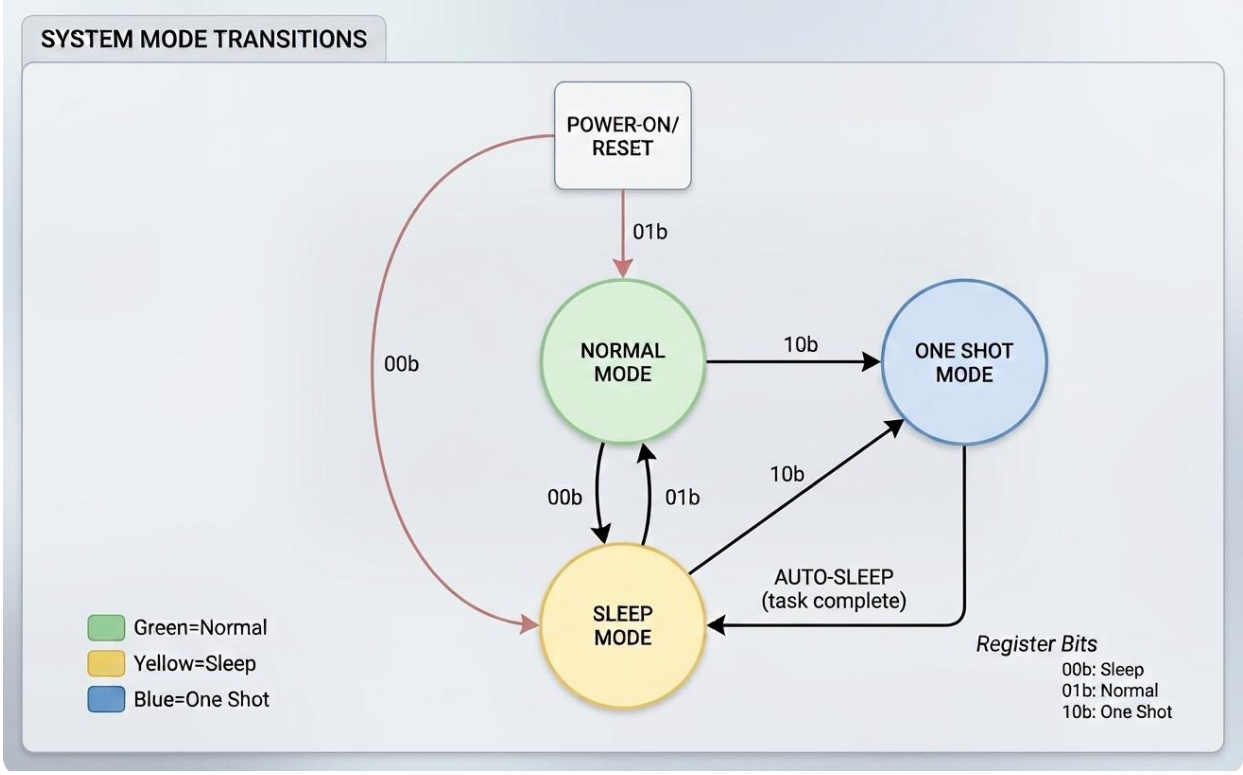
FUNCTION DESPCRIPTION

The sensor support three operation modes to measure the sensor signal according to the application requirement: Normal mode, Sleep mode and One shot mode.

Normal Mode, When powered on,the sensor automatically enters Normal Mode(Default mode). If switching from another mode to Normal Mode, it can be enabled by writing 01b to the MODE register (0x02[1:0]).The pressure and temperature sensor signals output measurement data cyclically at a predetermined frequency(Normally the standby time was pre-configured with 0mS).

One Shot Mode:It can be enabled by writing 10b to the MODE register (0x02[1:0]). The user can specify whether to measure the temperature or pressure signal by clearing or setting the measurement_ctrl bit (0x01[2]). After completing a single measurement,the sensor enters Sleep mode to await the next command.

Sleep Mode is lowest power consumption mode. The above mode can change between each other after the present command execution finish.



* For One Shot Mode, set ONE_PT(0x01.[2]) for pressure or temperature measurement before set MODE register (0x02[1:0])

PRESSURE CALCULATION

Read the pressure data from 0x04 to 0x06 register, the calculation formula for the actual pressure is as below:

Sum = (0x04 value * 2^16 + 0x05 value * 2^8 + 0x06 value),

If sum < 8388608, *P=sum / 2^21*(PMAX - PMIN)* (Unit: Pa)

If sum ⩾8388608, *P=(sum-2^24)/2^21*(PMAX - PMIN)* (Unit: Pa)

Range	PMIN	PMAX	Example(Unit: Pa)
Positive Pressure	0	Upper Range	0~100kPa: PMIN=0, PMAX:100000
Negative Pressure	Lower Range	0	-20~0kPa: PMIN= -20000, PMAX:0
N to P Pressure	Lower Range	Upper Range	-40~40kPa: PMIN=-40000, PMAX:40000

TEMPERATURE CALCULATION

Read the pressure data from 0x07 and 0x08 register, the calculation formula for the actual temperature is as below:

Final_T = (Inter_T - Byte1) / 2 ^ Shift_N + 25

The specific calculation steps are as follows:

1, Inter_T = RAW_T - 65536 (RAW_T > 32768); otherwise, Inter_T = RAW_T, where
RAW_T = 0x07 value * 2^8 + 0x08 value

2, Byte1 is calculated from the 0x20 register value, where
Bits [6:0] are used as the exponent of 2,
Bit [7] is the sign bit, when bit [7] = 1, Byte1 is a negative value, when bit [7] = 0, Byte1 is a positive value.
Assumed the value of 0x20 register is 0x8D (i.e. 10001101), according to the above description, the exponent is 13, the sign bit is negative, so the value of Byte1 is -(2^13), i.e. -8192.

3, Shift_N = Byte2/10, where Byte2 is the 0x21 register value. Assumed the value of 0x21 register is 0x46, then Shift_N=70/10=7.

PACKING INFORMATION

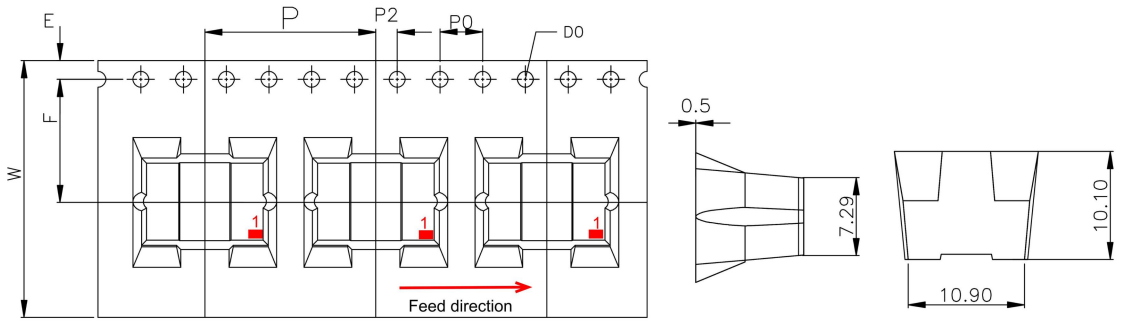
Tube Packing

Packing	Plastic Tube	Inner Box	Note
Quantity	70PCS per tube	2800pcs(40pcs tube)	Anti-static bag

Note:The sensor should be stored in an ESD protective container before using them.

Tape&Reel

Packing	Tape&Reel	Outer Box	Note
Quantity	400PCS per reel	6400pcs(16pcs tube)	Anti-static bag



W	24.00±0.30	P	12.00±0.10	A0	10.9±0.10	B0	7.29±0.10
S0	———	P0	4.00±0.10	A1		B1	
E	1.75±0.10	P2	2.00±0.10	A2		B2	
F	11.5±0.10	D0	ø1.50$^{+0.10}_{0}$	K0	10.1±0.10		
T	0.50±0.05	D1		K1			

Note: 1, Unless otherwise requested, the default packing method is Tube Packing.

2, The packing size may be not quite same with above for other different quantity and samples.

OVERALL NOTES

Unless otherwise specified, following notes are general attention or presentation for all products from CFSensor.

Mounting

The following steps is for transmitting the air pressure to sensor after sensor soldering on PCB.

▼ For some sensors that come with inlet tube, select the flexible pipe to suit the pressure inlet that is firm enough to prevent the pressure leaks.

▼ Atmosphere hole (for Gauge type sensors) and Inlet pipe/hole can't be blocked with gel or glue etc,..

▼ Avoiding excessive external force operation

Soldering

Due to its small size, the thermal capacity of the pressure sensor is low. Therefore, take steps to minimize the effects of external heat. Damage and changes to characteristics may occur due to heat deformation. Use a non-corrosive resin type of flux. Since the pressure sensor is exposed to the atmosphere, do not allow flux to enter inside.

▼ Manual soldering

⊙Raise the temperature of the soldering tip between 260 and 300°C/500 and 572°F (30 W) and solder within 5 seconds.

⊙The sensor output may vary if the load is applied on the terminal during soldering.

⊙Keep the soldering tip clean.

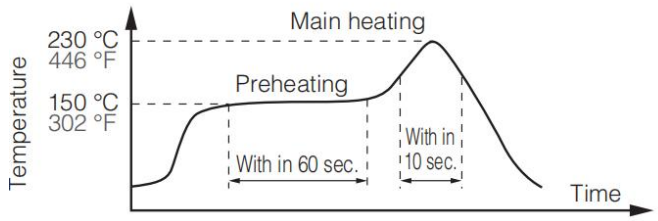
▼DIP soldering (DIP Terminal)

⊙Keep the temperature of the DIP solder tank below 260°C/500 and solder within 5 seconds.

⊙To avoid heat deformation, do not perform DIP soldering when mounting on the PCB which has a small thermal capacity.

▼Reflow soldering (SMD Terminal)

⊙The recommended reflow temperature profile conditions are given below.



⊙Self alignment may not always work as expected, therefore, please carefully note the position of the terminals and pattern.

⊙The temperature of the profile is assumed to be a value measured with the PCB of the terminal neighborhood.

⊙Please evaluate solderability under the actual mounting conditions since welding and deformation of the pressure inlet port may occur due to heat stress depending on equipments or conditions.

▼ Rework soldering

⊙Complete rework at a time.

⊙Use a flattened soldering tip when performing rework on the solder bridge. Do not add the flux.

⊙Keep the soldering tip below the temperature described in the specifications.

▼ Avoid drop and rough handling as excessive force may deform the terminal and damage soldering characteristics.

▼ Keep the circuit board warpage within 0.05 mm of the full width of the sensor.

▼ After soldering, do not apply stress on the soldered part when cutting or bending the circuit board.

▼ Prevent human hands or metal pieces from contacting with the sensor terminal. Such contact may cause anomalous outlets as the terminal is exposed to the atmosphere.

▼ After soldering, prevent chemical agents from adhering to the sensor when applying coating to avoid insulation deterioration of the circuit board.

Connecting

▼ Correctly wire as in the connection diagram. Reverse connection may damage the product and degrade the performance.

▼ Do not use idle terminals(N/C) to prevent damages to the sensor.

Cleaning

▼ Since the pressure sensor is exposed to the atmosphere, do not allow cleaning fluid to enter inside from atmosphere hole (for Gauge type sensors) and inlet pipe.

▼ Avoid ultrasonic cleaning since this may cause breaks or disconnections in the wiring.

Environment

▼Please avoid using or storing the pressure sensor in a place exposed to corrosive gases (such as the gases given off by organic solvents, sulfurous acid gas, hydrogen sulfides, etc.) which will adversely affect the performance of the pressure sensor chip.

▼Since this pressure sensor itself does not have a water-proof construction(even measurable medium can be liquid), please do not use the sensor in a location where it may be sprayed with water, etc.

▼Avoid using the pressure sensors in an environment where condensation may form. Furthermore, its output may fluctuate if any moisture adhering to it freezes.

▼The pressure sensor is constructed in such a way that its output will fluctuate when it is exposed to light. Especially when pressure is to be applied by means of a transparent tube, take steps to prevent the pressure sensor chip from being exposed to light.

▼Avoid using pressure sensor where it will be susceptible to ultrasonic or other high-frequency vibration.

▼Keeping the sensors sealed in static shielding bags with an oxygen-free condition and use the sensor as soon as possible once unfold the package, because the sensors' PINs may be oxidated a bit under atmosphere environment(slight oxidation wouldn't affect soldering and performance)

More Precautions

▼That using the wrong pressure range or mounting method may result in accidents.

▼The only direct pressure medium you can use is non-corrosive gas or air as illuminated above(Note: some sensors are compatible with liquid media). The use of other media, in particular, corrosive gases and liquid (organic solvent based, sulfurous acid based, and hydrogen sulfide based, etc.) or contains foreign substances will cause malfunction and damage. Please do not use them and check with CFSensor.

▼The pressure sensor is positioned inside the pressure inlet. Never poke wires or other foreign matter through the pressure inlet since they may damage the sensor or block the inlet. Avoid use when the atmospheric pressure inlet(only for Gauge type pressure sensor) is blocked.

▼Use an operating pressure which is within the rated pressure range. Using a pressure beyond this range may cause damage.

▼Since static charge can damage the pressure sensor, bear in mind the following handling precautions.

⊙When storing the pressure sensor, use a conductive material to short the pins or wrap the entire sensor in aluminum foil. Common plastic containers should not be used to store or transport the sensor since they readily become charged.

⊙When using the pressure sensor, all the charged articles on the bench surface and the work personnel should be grounded so that any ambient static will be safely discharged.

▼Based on the pressure involved, give due consideration to the securing of the pressure sensor.

【 SAFETY NOTES 】

Using these sensors products may malfunction due to external interference and surges, therefore, please confirm the performance and quality in actual use. Just in case, please make a safety design on the device (fuse, circuit breaker, such as the installation of protection circuits, multiple devices, etc.), so it would not harm life, body, property, etc even a malfunction occurs. To prevent injuries and accidents, please be sure to observe the following items:

- The driving current and voltage should be used below the rated value.
- Please follow the terminal connection diagram for wiring. Especially for the reverse connection of the power supply, it will cause an accident due to circuit damage such as heat, smoke, fire, etc.
- In order to ensure safety, especially for important uses, please be sure to consider double safety circuit configuration.
- Do not apply pressure above the maximum applied pressure. In addition, please be careful not to mix foreign matter into the pressure medium. Otherwise, the sensor will be discarded, or the media will blew out and cause an accident.
- Be careful when fixing the product and connecting the pressure inlet. Otherwise, accidents may occur due to sensor scattering and the blowing out of the media.
- If the sensor come with sharp PIN, please be careful not to hurt your body when using it.

【 WARRANTY 】

The information in this sheet has been carefully reviewed and is believed to be accurate; however, no responsibility is assumed for inaccuracies. Furthermore, this information does not convey to the purchaser of such devices any license under the patent rights to the manufacturer. CFSensor reserves the right to make changes without further notice to any product herein. CFSensor makes no warranty, representation or guarantee regarding the suitability of its product for any particular purpose, nor does CFSensor assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation consequential or incidental damages. Typical parameters can and do vary in different applications. All operating parameters must be validated for each customer application by customer's technical experts. CFSensor does not convey any license under its patent rights nor the rights of others.

【 CONTACT 】

CFSensor
22F/14Bldg High-Tech Park High-Tech Area Wuhu P.R.C.241000
Tel/Fax:+86 18226771331 Email:INFO@CFSensor.com

North America II Europe II Southeast Asia II Middle East II Latin America

IIC Example Code (8051 Platform)

```
#include <reg52.h>
#include <math.h>
#include <stdio.h>

/*=========================================================================*/
/*                      Configuration Macros                               */
/*=========================================================================*/

/* I2C timing - adjust based on your crystal frequency */
/* For 11.0592MHz: DELAY_TIME = 600 (about 5us per loop) */
/* For 12MHz:      DELAY_TIME = 550 (about 5us per loop) */
#define DELAY_TIME      600

/* Sensor I2C address (7-bit) */
#define SENSOR_ADDR     0x58

/* I2C read/write addresses (8-bit) */
#define I2C_WRITE_ADDR  (SENSOR_ADDR << 1)      /* 0xB0 */
#define I2C_READ_ADDR   ((SENSOR_ADDR << 1) | 1) /* 0xB1 */

/* Sensor registers */
#define REG_CTRL        0x01    /* Control register */
#define REG_DATA_START  0x04    /* Data start address (5 bytes) */
#define REG_EOF         0x20    /* Temperature offset register */
#define REG_SHIFT       0x21    /* Temperature gain register */

/*=========================================================================*/
/*                      Pressure Range Configuration                       */
/*=========================================================================*/
/*
 * NOTE: Modify PMIN and PMAX according to your specific sensor model
 *
 * Common Ranges for I2C ADD 0x58 Sensors (unit: Pa):
 *   0 ~ 0.5kPa:     PMIN = 0.0f,       PMAX = 500.0f
 *   0 ~ 5kPa:       PMIN = 0.0f,       PMAX = 5000.0f
 *   0 ~ 1000kPa:    PMIN = 0.0f,       PMAX = 1000000.0f
 *   -1 ~ 1kPa:      PMIN = -1000.0f,   PMAX = 1000.0f
 *   -5 ~ 5kPa:      PMIN = -5000.0f,   PMAX = 5000.0f
 *   -100 ~ 100kPa:  PMIN = -100000.0f, PMAX = 100000.0f
 */
```

```
#define PMIN           -100000.0f      /* Lower limit: -100 kPa */
#define PMAX           100000.0f       /* Upper limit: 100 kPa */

/*=========================================================================*/
/*                        Type Definitions                                 */
/*=========================================================================*/
#define TRUE           1
#define FALSE          0
#define uchar          unsigned char
#define uint           unsigned int

/*=========================================================================*/
/*                        Pin Definitions                                  */
/*=========================================================================*/
sbit SCL = P1 ^ 7;      /* I2C clock line */
sbit SDA = P1 ^ 6;      /* I2C data line */

/*=========================================================================*/
/*                        Global Variables                                 */
/*=========================================================================*/
bit g_i2c_error;        /* I2C error flag */

/*=========================================================================*/
/*                        Delay Functions                                  */
/*=========================================================================*/

/**
  * @brief  Microsecond delay (software loop)
  * @param  t: Delay count
  * @note   Adjust DELAY_TIME based on your crystal frequency
  */
void DELAY(uint t)
{
    while (t != 0)
        t--;
}

/**
  * @brief  Millisecond delay
  * @param  x: Delay time in milliseconds
  * @note   Calibrated for 11.0592MHz/12MHz
  */
```

```
void Delay_xms(uint x)
{
    uint i, j;
    for (i = 0; i < x; i++)
        for (j = 0; j < 112; j++)
            ;
}

/*=============================================================================*/
/*                        I2C Timing Functions                                 */
/*=============================================================================*/

/**
  * @brief  Generate I2C START condition
  * @note   SDA falling edge while SCL is high
  */
void I2C_Start(void)
{
    SDA = 1;
    DELAY(DELAY_TIME);
    SCL = 1;
    DELAY(DELAY_TIME);
    SDA = 0;
    DELAY(DELAY_TIME);
    SCL = 0;
    DELAY(DELAY_TIME);
}

/**
  * @brief  Generate I2C STOP condition
  * @note   SDA rising edge while SCL is high
  */
void I2C_Stop(void)
{
    SDA = 0;
    DELAY(DELAY_TIME);
    SCL = 1;
    DELAY(DELAY_TIME);
    SDA = 1;
    DELAY(DELAY_TIME);
    SCL = 0;
    DELAY(DELAY_TIME);
```

```
}

/**
  * @brief  Send bit '0' on I2C bus
  */
void SEND_0(void)
{
    SDA = 0;
    DELAY(DELAY_TIME);
    SCL = 1;
    DELAY(DELAY_TIME);
    SCL = 0;
    DELAY(DELAY_TIME);
}

/**
  * @brief  Send bit '1' on I2C bus
  */
void SEND_1(void)
{
    SDA = 1;
    DELAY(DELAY_TIME);
    SCL = 1;
    DELAY(DELAY_TIME);
    SCL = 0;
    DELAY(DELAY_TIME);
}

/**
  * @brief  Check slave's ACK signal
  * @retval TRUE: ACK received, FALSE: NACK received
  */
bit Check_Acknowledge(void)
{
    bit ack_status = 0;
    uint timeout = 1000;    /* Timeout counter */

    SDA = 1;
    DELAY(DELAY_TIME);
    SCL = 1;
    DELAY(DELAY_TIME / 2);
```

```
    /* Wait for SDA low (ACK) with timeout */
    while (SDA && timeout) {
        timeout--;
    }

    ack_status = (SDA == 0) ? TRUE : FALSE;
    DELAY(DELAY_TIME / 2);
    SCL = 0;
    DELAY(DELAY_TIME);

    if (timeout == 0) {
        g_i2c_error = 1;    /* Set error flag on timeout */
    }

    return ack_status;
}

/**
  * @brief  Send NACK signal (for ending read sequence)
  */
void Send_NACK(void)
{
    SDA = 1;
    DELAY(DELAY_TIME);
    SCL = 1;
    DELAY(DELAY_TIME);
    SCL = 0;
    DELAY(DELAY_TIME);
}

/**
  * @brief  Write one byte to I2C bus
  * @param  b: Byte to send (MSB first)
  */
void WriteI2CByte(uchar b)
{
    uchar i;
    for (i = 0; i < 8; i++) {
        if (b & 0x80)
            SEND_1();
        else
            SEND_0();
```

```
        b <<= 1;
    }
}

/**
  * @brief  Read one byte from I2C bus
  * @retval Byte received (MSB first)
  */
uchar ReadI2CByte(void)
{
    uchar i;
    uchar b = 0;

    SDA = 1;    /* Release SDA for slave to drive */

    for (i = 0; i < 8; i++) {
        b <<= 1;
        SCL = 1;
        DELAY(DELAY_TIME);
        if (SDA) {
            b |= 0x01;
        }
        SCL = 0;
        DELAY(DELAY_TIME);
    }

    return b;
}

/*============================================================================*/
/*                        High-Level I2C Operations                           */
/*============================================================================*/

/**
  * @brief  Write one byte to a register
  * @param  addr: Register address
  * @param  data: Data byte to write
  * @retval TRUE: Success, FALSE: Failure
  */
bit Write_One_Byte(uchar addr, uchar data)
{
    g_i2c_error = 0;
```

```
    I2C_Start();
    WriteI2CByte(I2C_WRITE_ADDR);
    if (!Check_Acknowledge()) {
        I2C_Stop();
        return FALSE;
    }

    WriteI2CByte(addr);
    if (!Check_Acknowledge()) {
        I2C_Stop();
        return FALSE;
    }

    WriteI2CByte(data);
    if (!Check_Acknowledge()) {
        I2C_Stop();
        return FALSE;
    }

    I2C_Stop();
    return TRUE;
}

/**
  * @brief  Read one byte from a register
  * @param  addr: Register address
  * @retval Data byte read, or 0xFF on error
  */
uchar Read_One_Byte(uchar addr)
{
    uchar data;

    g_i2c_error = 0;

    /* Write register address */
    I2C_Start();
    WriteI2CByte(I2C_WRITE_ADDR);
    if (!Check_Acknowledge()) {
        I2C_Stop();
        return 0xFF;
    }
```

```
    WriteI2CByte(addr);
    if (!Check_Acknowledge()) {
        I2C_Stop();
        return 0xFF;
    }

    /* Restart and read data */
    I2C_Start();
    WriteI2CByte(I2C_READ_ADDR);
    if (!Check_Acknowledge()) {
        I2C_Stop();
        return 0xFF;
    }

    data = ReadI2CByte();

    /* Send NACK to end read sequence */
    Send_NACK();

    I2C_Stop();
    return data;
}

/**
  * @brief  Read 5 consecutive bytes from 0x04 (3 pressure + 2 temp)
  * @param  buf: Buffer to store data (must have 5 bytes)
  * @retval TRUE: Success, FALSE: Failure
  */
bit Read_5_Bytes(uchar *buf)
{
    uchar i;

    g_i2c_error = 0;

    /* Write register address 0x04 */
    I2C_Start();
    WriteI2CByte(I2C_WRITE_ADDR);
    if (!Check_Acknowledge()) {
        I2C_Stop();
        return FALSE;
    }
```

```
    WriteI2CByte(0x04);
    if (!Check_Acknowledge()) {
        I2C_Stop();
        return FALSE;
    }

    /* Restart and read 5 bytes */
    I2C_Start();
    WriteI2CByte(I2C_READ_ADDR);
    if (!Check_Acknowledge()) {
        I2C_Stop();
        return FALSE;
    }

    for (i = 0; i < 5; i++) {
        buf[i] = ReadI2CByte();
        if (i < 4) {
            /* Send ACK for all but last byte */
            SDA = 0;
            DELAY(DELAY_TIME);
            SCL = 1;
            DELAY(DELAY_TIME);
            SCL = 0;
            DELAY(DELAY_TIME);
            SDA = 1;
        }
    }

    /* Send NACK after last byte */
    Send_NACK();

    I2C_Stop();
    return TRUE;
}

/*=========================================================================*/
/*                      Sensor Data Processing                             */
/*=========================================================================*/

/**
  * @brief  Get temperature calibration coefficients
```

```
  * @param  eof_out: Pointer to store EOFOut
  * @param  shift_n: Pointer to store shiftN
  * @retval TRUE: Success, FALSE: Failure
  */
bit Get_Calibration_Coefficients(int *eof_out, float *shift_n)
{
    uchar byte1, byte2;

    byte1 = Read_One_Byte(REG_EOF);
    if (byte1 == 0xFF) return FALSE;

    byte2 = Read_One_Byte(REG_SHIFT);
    if (byte2 == 0xFF) return FALSE;

    /* Decode EOFOut from register 0x20 */
    switch (byte1) {
        case 0x0C:  *eof_out = 4096;   break;
        case 0x8C:  *eof_out = -4096;  break;
        case 0x0D:  *eof_out = 8192;   break;
        case 0x8D:  *eof_out = -8192;  break;
        case 0x0E:  *eof_out = 16384;  break;
        case 0x8E:  *eof_out = -16384; break;
        default:    *eof_out = 0;      break;
    }

    /* Calculate shiftN from register 0x21 */
    *shift_n = pow(2.0f, (float)byte2 / 10.0f);

    return TRUE;
}

/*=============================================================================*/
/*                        Main Function                                        */
/*=============================================================================*/

void main(void)
{
    uchar raw_data[5];
    long int pressure_ad;
    long int temperature_ad;
    float pressure;
    float temperature;
```

```
int eof_out;
float shift_n;

/* Wait for system to stabilize */
Delay_xms(1000);

while (1) {
    /* Step 1: Start measurement */
    if (!Write_One_Byte(REG_CTRL, 0x01)) {
        printf("Error: Write control failed\r\n");
        Delay_xms(1000);
        continue;
    }
    Delay_xms(20);

    /* Step 2: Read 5 bytes from 0x04 */
    if (!Read_5_Bytes(raw_data)) {
        printf("Error: Read data failed\r\n");
        Delay_xms(1000);
        continue;
    }

    /* Step 3: Combine 24-bit pressure data */
    pressure_ad = ((long int)raw_data[0] << 16) |
                  ((long int)raw_data[1] << 8)  |
                  (long int)raw_data[2];

    /* Step 4: Sign extension for 24-bit pressure */
    if (pressure_ad & 0x800000) {
        pressure_ad |= 0xFF000000;  /* Extend sign bit */
    }

    /* Step 5: Combine 16-bit temperature data */
    temperature_ad = ((long int)raw_data[3] << 8) |
                     (long int)raw_data[4];

    /* Step 6: Convert temperature to signed 16-bit */
    if (temperature_ad >= 32768) {
        temperature_ad -= 65536;
    }

    /* Step 7: Get calibration coefficients */
```

```
        if (!Get_Calibration_Coefficients(&eof_out, &shift_n)) {
            printf("Error: Read calibration failed\r\n");
            Delay_xms(1000);
            continue;
        }

        /* Step 8: Calculate pressure (Pa) */
        if (pressure_ad < 0) {
            pressure = (float)pressure_ad / 2097152.0f * (PMAX - PMIN);
        } else {
            pressure = (float)pressure_ad / 2097152.0f * (PMAX - PMIN);
        }

        /* Step 9: Calculate temperature (℃) */
        temperature = ((float)(temperature_ad - eof_out) / shift_n) + 25.0f;

        /* Step 10: Output results */
        printf("Pressure_AD: %ld (0x%06lX)\r\n", pressure_ad, pressure_ad);
        printf("Temperature_AD: %ld (0x%04lX)\r\n", temperature_ad, temperature_ad);
        printf("EOFOut: %d, shiftN: %.2f\r\n", eof_out, shift_n);
        printf("Pressure: %.2f Pa (%.2f kPa)\r\n", pressure, pressure / 1000.0f);
        printf("Temperature: %.2f C\r\n\r\n", temperature);

        Delay_xms(1000);
    }
}
```